Hochschule Offenburg
University of Applied Sciences

M+I Medien und Informationswesen

Hochschule Offenburg

Medien und Informationswesen

Bachelor-Thesis

Konzeptionierung und Realisierung einer skriptbasierten Lösung zur automatisierten Aufgabenbewältigung im Umfeld einer Webagentur

Sommersemester 2014

arteria GmbH

vorgelegt von
Walter Renner

Betreut von
Prof. Dr. Tom Rüdebusch
Dipl. Ing. Philippe O. Wagner

Inhaltsverzeichnis

1 Einleitung

1.1 Aufgabenstellung und Ziele der Bachelor-Thesis

Oft anfallende Prozesse und Aufgaben im IT-Umfeld kosten viel Zeit und personelle Ressourcen. Zudem können bei einer manuellen Ausführung dieser Aufgaben Fehler entstehen. Die Effizienz und die Wettbewerbsfähigkeit derer Unternehmen sinkt, die diese Aufgaben nicht unter Kontrolle haben. Ziel ist es wiederkehrende Aufgaben mit wenig Aufwand zu automatisieren und so das Fehlerpotential zu vermindern.

Eine besondere Herausforderung ergibt sich hierbei in der Aggregation der diversen verteilten Konten, wie Mitarbeiterlaptop, Server und online genutzte Dienste.

In dieser Arbeit sollen zunächst Anforderungen an die zu entwickelnde Lösung festgelegt werden. Die erarbeiteten Anforderungen sollen anschließend den vorhandenen Lösungen gegenübergestellt werden, um diese bewerten zu können. Anschließend erfolgt die Implementierung einer neuen oder Anpassung und Erweiterung einer vorhandenen Lösung.

1.2 Aufbau der Thesis

Im ersten Kapitel werden Aufgabenstellung und Inhalte der Bachelor-Thesis dargestellt.

In Kapitel zwei werden alle, für diese Arbeit relevanten, Begriffe und Konzepte definiert und erläutert.

Im dritten Kapitel werden drei bestehende Systeme anhand konkreter Aufgaben verglichen. Für diesen Vergleich werden zunächst die Kriterien definiert. Mit den Ergebnissen der Untersuchung wird entschieden, ob sich eine dieser Anwendungen für den Einsatz im Unternehmen eignet.

Thema des vierten Kapitels sind die Anforderungen, die an das neue System gestellt werden. Insbesondere werden hier die Erkentnisse des hervorgegangenen Vergleichs refliektiert und behandelt.

Die Konzeption des zu entwickelnden Systems wird im fünften Kapitel erläutert. Hierfür werden zunächst die wichtigsten Ansätze dargelegt und anschießend die verwendeten Technologien beschrieben.

Das sechste Kapitel behandelt die Implemtierung des neuen Systems. Besonderer Fokus liegt hier auf den Herausforderungen und Schwierigkeiten in der Entwicklungsphase.

Im letzten Kapitel wird das entwickelte System den Anforderungen gegenübergestellt, die im vierten Kapitel definiert wurden. Weiterhin werden in diesem Kapitel bekannte Schwächen des Systems aufgezeigt. Anschließend folgt ein Ausblick, der darlegt mit welchen Schritten das System in Zukunft optimiert und weiterentwickelt werden kann.

2 Grundlagen

2.1 Warum Automatisierung?

Mit der Automatisierung von IT-Aufgaben erzielt man sowohl operative als auch strategische Vorteile für das Unternehmen.

Zum einen wird das IT-Personal in physischer und psychischer Hinsicht entlastet, was dazu führt, dass mehr Zeit für die Entwicklung geschaffen wird. Andererseits können so auch Flüchtigkeitsfehler bei der manuellen Ausführung von IT-Aufgaben unterbunden werden.

Ein weiterer Punkt ist die schnellere Durchführung von IT-Prozessen. Dies ist voralem dann ausschlaggebend, wenn viele Interaktionen durch Personal notwendig sind. Gerade in diesen Aufgaben lässt sich ein hoher Automatisierungsgrad umsetzen.

2.2 Workflow-Management

Workflow-Management ist die Koordination und Kontrolle von Geschäftsprozessen. Geschäftsprozesse sind eine Menge logisch verknüpfter Aufgaben die ausfegührt werden, um ein Geschäftsziel zu erreichen.

Zusammengefasst erfüllt das zum Workflow-Management folgende Aufgaben [1]:

- Spezifikation von Geschäftsprozessen
- Verwaltung vor Ressourcen (personell und materiell)
- Implementierung der modellierten Geschäftsprozesse

- Ausührung der Geschäftsprozesse
- Optimierung durch Analyse und Reorganisation

Diese Aufgaben werden meist mithilfe von IT unterstützt.

Workflow-Management beschreibt also: **Wer macht was wie und womit?**

Die Besonderheit dieser Arbeit besteht darin, dass Workflows ohne jegliche Benutzerinteraktion durchgeführt werden müssen. Es soll lediglich eine Erfolgskontrolle durch das Personal erfolgen, indem Sie die Protokolle der Durchführung analysieren.

2.3 Software-Konfigurationsmanagement

Das Konfigurationsmanagement stellt sicher, dass ein Produkt und die darin enthaltenen Konfigurationseinheiten, die dieses Produkt bilden, vollständig reproduzierbar sind. [2]

Das Software-Konfigurationsmanagement ist eine Spezialisierung des Konfigurationsmanagements auf alle Aktivitäten im Bereich der Software-Entwicklung.

Ziele des Software-Konfigurationsmanagement sind: [3]

- Änderungen kontrollieren
- Qualität sicherstellen
- Produktivität steigern
- Transparenz verbessern

2.4 Kontinuierliche Integration

Die kontinuierliche Integration ist Teil des Software-Konfigurationmanagement. Im Kern geht es darum, die Integration aller Codeänderungen nicht mehr als eine ein-

malige Phase am Ende eines Entwicklungsprojekts zu betrachten. Vielmehr soll diese Endmontage (eng. Build) einer Software viel häufiger erfolgen, typischerweise sogar mehrfach am Tag. [4]

Dabei werden drei Arten von Builds (Automatisierungstypen) unterschieden: [5]

- Befehlsgesteuerte Automatisierung: Erfolgt jedes mal nachdem ein Befehl ausgeführt wurde. Beispielsweise das Ausführen eines Skripts in einer Kommandozeile
- Zeitgesteuerte Automatisierung: Der Build erfolgt zu einer bestimmten Uhrzeit in regelmäßigen Abständen. Beispielsweise mithilfe von Cronjobs.
- Ereignisgesteuerte Automatisierung: Ein Build wird dann ausgeführt, wenn ein definiertes Ereignis eingetreten ist. Beispeilsweise bei jeder Änderung am Quellcode.

Die einfachste Art kontinuierlich zu Integrieren ist die Verwendung von Build-Skripten. Dabei werden eine Reihe von Aufgaben in einer konsistenen und wiederholbaren Weise ausgeführt.

2.5 Zusammenfassung

Build-Skripte stellen die einfachste Art der Kontinuierlichen Integration dar. Bei Änderungen an Systemen oder der IT-Infrastruktur müssen diese oft angepasst werden. Daher ist der Automatisierungsgrad entsprechend gering. Weil aber im Unternehmen bereits mit Build-Skripten gearbeitet wird, würde es sich anbieten diese weiterhin benutzen zu können.

Die in dieser Arbeit gesuchte Lösung ist also eine Anwendung die mindestens folgende Voraussetzungen erfüllt:

- Das Verwalten und die Ausführung von Workflows
- Möglichkeit Kontinuierliche Integration im Unternehmen einzuführen
- Wiederverwendung der bereits verwendeten Build-Skripte

3 Untersuchung bestehender Anwendungen

Die in Kapitel 2.5 festgelegten Voraussetzungen bilden die Grundlage für die Recherche nach vorhandenen Systemen. Um möglichst unterschiedliche Anwendungen zu testen und einen guten Überblick zu erhalten, werden aus zunächst circa zehn Kandidaten drei ausgewählt.

Kriterien und Beispielaufgaben für die Untersuchung werden vor der Durchführung festgelegt und im folgenden Kapitel beschrieben.

3.1 Ausgewählte Lösungen

3.1.1 Jenkins

Jenkins ist wohl das am weitesten verbreitete System zur Kontinuierlichen Integration. Es wurde von Kohsuke Kawaguchi entwickelt während er bei Sun Microsystems angestellt war. Zunächst wurde es unter dem Namen Hudson verbreitet. Später verließ Kawaguchi das Unternehmen. Da er die Namensrechte an Hudson nicht besitzt, benannte er seinen sog. Fork in Jenkins um. [6]

Jenkins ist ein Open Source Projekt und ist unter der MIT-Lizenz verfügbar. Es ist Java geschrieben und bietet weit über 300 Plugins zur Erweiterung des Basissystems. Nach der Installation kann es über ein Webinterface konfiguriert und verwaltet werden.

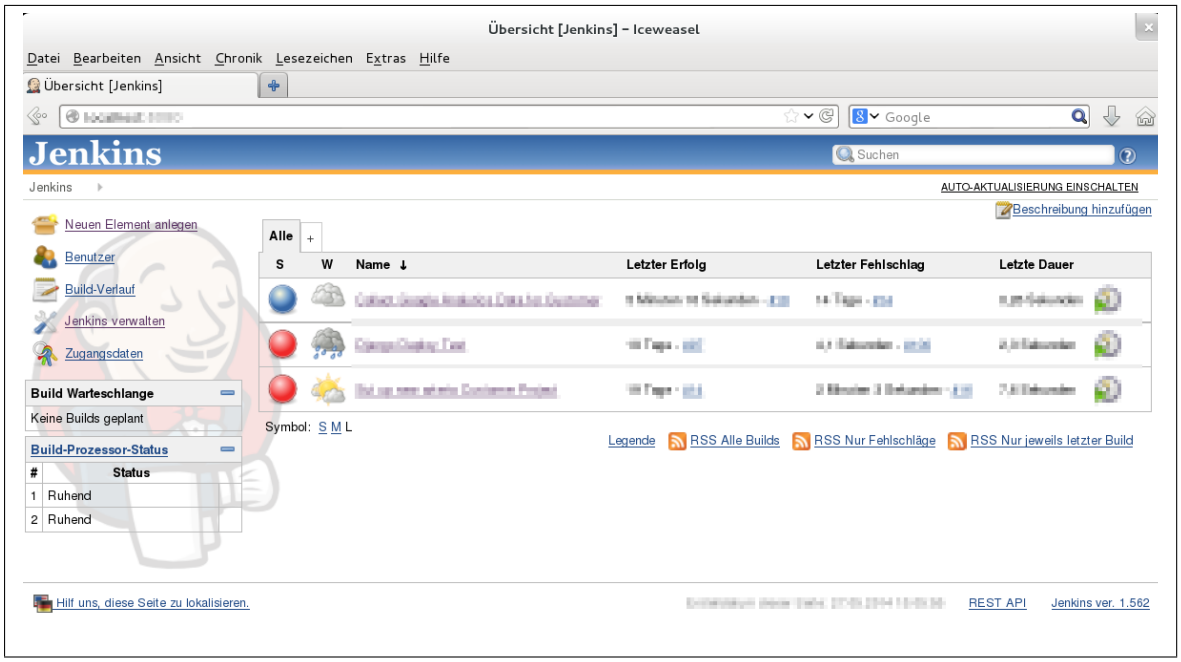


Abbildung 3.1: Die Startseite von Jenkins

3.1.2 Buildbot

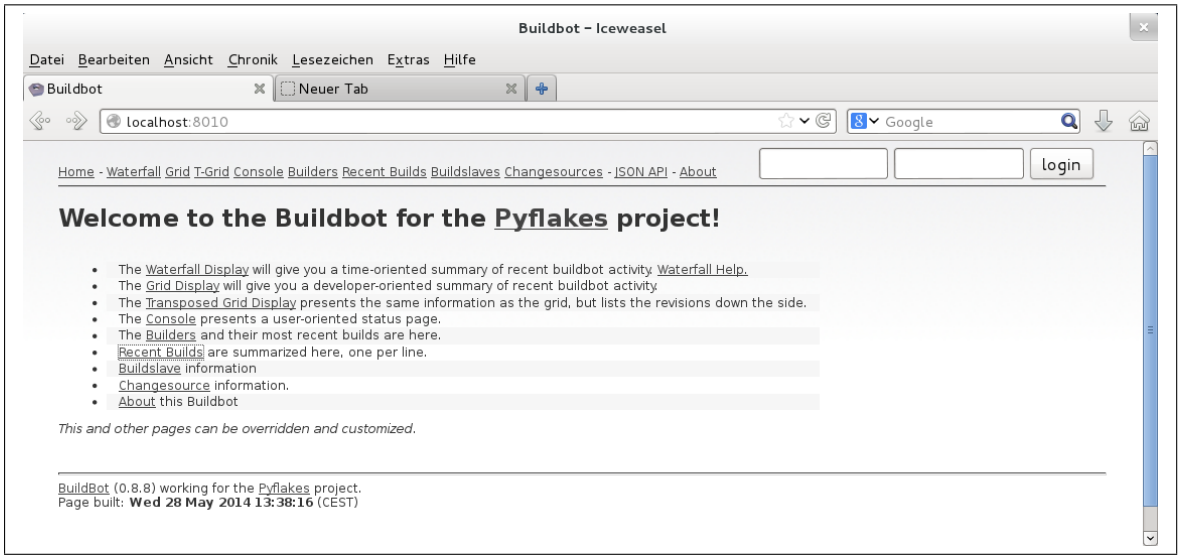


Abbildung 3.2: Die Startseite von Buildbot

Im Kern ist Buildbot ein Aufgaben-Verarbeitungs-System. Es können Aufgaben erstellt werden, die dann ausgeführt werden, wenn ein Ereignis auftritt. [7] Buildbot erstellt für jede Durchführung einen Bericht mit Benutzerfeedback und Fehlermeldungen, die während der Durchführung aufgetreten sind.

Eine Buildbot Installation hat einen oder mehrere Master und ein Reihe von Slaves.

Die Master beobachten Quellcode-Repositories auf Veränderungen und reagieren auf diese, indem sie die Slaves anstoßen. Auf den Slaves wird dann eine Menge an Aufgaben durchgeführt und die Ergebnisse werden zurück an dem Master gesendet. Dieser stellt diese in einer HTML-Datei grafisch dar. Master und Slaves müssen nicht zwingend auf unterschiedlichen Systemen installiert werden.

Um Buildbot zu konfigurieren, erstellt man auf dem Master ein Konfigurations-Skript in Python. Dies ist umständlich, hat aber den Vorteil, dass man alle Befehle einer Programmiersprache nutzen kann.

Das Framework selbst ist ebenfalls in Python geschrieben und funktioniert deshalb auf allen gängigen Betriebssystemen.

3.1.3 deploy.do

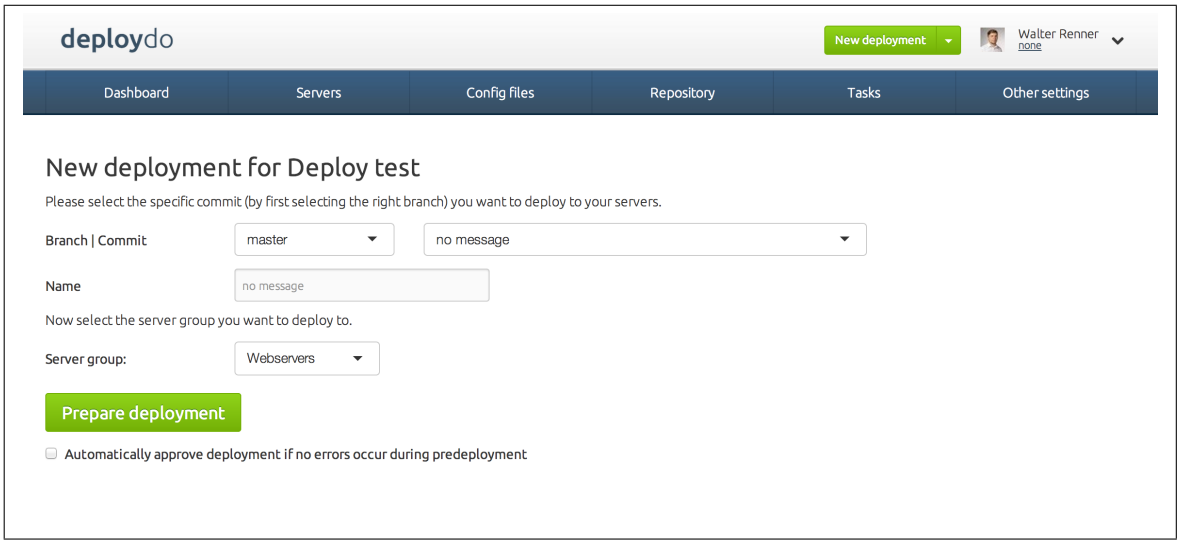


Abbildung 3.3: Die Startseite von deploy.do

Um auch eine Cloud Lösung zu testen, wurde deploy.do als Kandidat gewählt.

Deploy.do ist ein SaaS (Software as a Service) Tool zur Installation eines Git-Repositories auf einem oder mehreren Servern. Die hauptsächlichen Merkmale sind der Linie-für-Linie Vergleich des Quellcodes, bevor dieser tatsächlich installiert wird. [8] deploy.do benutzt für seine Webanwendung die Skriptsprache PHP. Die Kommunikation zu den Servern wird mit SSH realisiert.

Oft anfallende Aufgaben, wie beispielsweise das Neustarten der Server, können mithilfe von Tasks schnell erledigt werden. Ein Nachteil einer solchen Lösung liegt

bereits vor dem Testen auf der Hand. Diese Dienste können nicht angepasst und erweitert werden. Das Unternehmen hinter deploy.do verlangt für das kleinste Paket 29Euro pro Monat und das teuerste Paket, mit 30 möglichen Projekten, zehn Servern und zehn Benutzern 199Euro pro Monat.

3.2 Untersuchung der gewählten Anwendungen

3.2.1 Untersuchungsbeschreibung

Die ausgewählten Applikationen sollen getestet werden, um festzustellen, ob diese sich für den täglichen Einsatz im Unternehmen eignen. Dazu wird in einer virtuellen Maschine eine Testumgebung eigerichtet, die dem Firmenserver ähnelt. Das Betriebssystem Debian, mit diversen virtuellen Umgebungen für Python, wird vor dem Testen installiert. Um die Untersuchungen, in einem der Realität entsprechendem Szenario durchzuführen, werden zwei Django-Projekte installiert und eingerichtet.

Die vereinfachte Verzeichnisstruktur sieht folgendesmaßen aus:

```
/home/<user>/workspace/<projekt1>_env/<projekt1>
                       <projekt2>_env/<projekt2>
```

Die Projekte verwenden jeweils unterschiedliche Django Versionen und in ihren Umgebungen unterschiedliche Python Pakete in teilweise unterschiedlichen Versionen.

Vor dem Test wird von der virtuellen Maschine ein Schnappschuss erstellt. Nach jedem Testdurchgang wird die VM mittels des Schnappschusses auf die Ausgangsbasis zurückgestellt, somit hat jedes Tool dieselben Voraussetzungen.

3.2.2 Untersuchungskriterien

Vor der Durchführung der Untersuchung werden zunächst Kriterien an die zu testenden Lösungen definiert.

Da im Unternehmen Mitarbeiter mit unterschiedlichem technischen Wissen angestellt sind soll die Anwendung einfach zu bedienen sein und eine **Ausführung der**

Aufgaben auch ohne technisches Know-How ermöglichen.

Ebenso sollte die Möglichkeit bestehen die **Arbeitsabläufe im Frontend zu erstellten und zu editieren**.

Weiterhin besitzt die gesuchte Lösung bestenfalls eine **Schnittstelle zur Erweiterung** der Applikation.

Da zwei der drei ausgesuchten Anwendungen quelloffen sind, ist eine starke **Community und eine weite Verbreitung** von Vorteil. Sehr wahrscheinlich kann man so auf zahlreiche Anleitungen, Tipps und Tricks der Benutzer zurückgreifen.

Zudem sollte die Möglichkeit bestehen Aufgaben **Ereignis- und zeitgesteuert auszuführen**.

Eine **Zugriffskontrolle pro Benutzer und Benutzergruppe** wäre ebenfalls wünschenswert, um so bestimmte Aufgaben nur von berechtigten Personen ausführen zu lassen.

Im Unternhemen wird auf Team bzw. Organisationsebene gearbeitet. Deshalb sollen **Teilaufgaben und Arbeitsabläufe dupliziert** und anderen Teams zur Verfügung gestellt werden können.

Da mit der Applikation auch teilweise sebsible Daten über ein unsicheres Medium übertragen werden, wird auch die **Sicherheit** des Systems betrachtet. Eine Verschlüsselung während des Transports gilt hier als wichtigstes Kriterium.

3.2.3 Festlegen der durchzuführenden Aufgaben

Aufgabe 1: Aufsetzen eines neuen Kundenprojekts

In der ersten Aufgabe soll ein neues Kundenprojekt auf der virtuellen Maschine installiert und eigerichtet werden.

Hierfür wird zunächst eine virtuelle Umgebung für Python erstellt. Danach werden mittels des Paket-Verwaltungs-Systems pip diverse Python Pakete installiert. Darunter auch Django, welches die Basis eines jeden Kundenprojektes im Unternehmen bildet. Anschließend wird ein Git-Repository initialisiert und mittels einer API auf Bitbucket geladen.

Diese Aufgabe wird bereits mit einem Skript im Unternehmen erledigt. Das Skript ist weitaus umfangreicher, für die Untersuchung wurden jedoch nur die wichtigsten Punkte übernommen.

Aufgabe 2: Deployment eines Codes auf einen Server

Die tägliche Aufgabe im Unternehmen ist die Entwicklung von Webapplikationen. Mehrmals am Tag werden Anwendungen aktualisiert und verbessert. Deshalb besteht die zweite Aufgabe der Untersuchung darin, den Quellcode eines Projektes auf der virtuellen Maschine zu installieren.

Ausgang dieser Aufgabe ist ein Git-Repository auf Bitbucket. Zunächst müssen also alle nötigen Daten heruntergeladen werden. Anschließend sollen die eventuell neu benötigten Python Pakete gemäß der requirementx.txt Datei installiert bzw. aktualisiert werden. Nach der erfolgreichen Installation wird die Datenbank aktualisiert beziehungsweise migriert. Zuletzt werden die in der Applikation implementierten Tests ausgeführt und der FastCGI Prozess neu gestartet, um die Änderungen auch dem nginx Webserver zur Verfügung zu stellen.

Aufgabe 3: Ansteuern einer API

In dieser Aufgabe soll eine API angesteuert werden. Nötig ist dies beispielsweise wenn Google-Analytics Daten eines Kunden abgerufen werden sollen. Diese Daten sollen per E-Mail an einen Mitarbeiter verschickt werden. Dieser erstellt einen monatlichen Bericht für den Kunden. Hier macht es Sinn, diese Aufgabe periodisch durchzuführen.

Zur Vereinfachung wurde hier der RSS-Feed des Firmenblogs ausgelesen und der Inhalt als E-Mail verschickt.

3.2.4 Durchführung mit Jenkins

Einrichtung

Zunächst muss Jenkins auf der virtuellen Maschine installiert werden. Dazu wird das Paket heruntergeladen und die Installationsroutine gestartet.

Danach kann die erstellte WAR-Datei (Web Application Archive) gestartet werden. Schon ist Jenkins im Browser über die Adresse http://localhost:8080/ erreichbar. Nun kann ein neues Projekt erstellt werden.

Aufgabe 1

Um die erste Aufgabe auszuführen, wird ein parametrisierter Build erstellt. Hier können Variablen vor der Ausführung eingegeben werden. Diese können dann in den folgenden Skripten als Umgebungsvariablen abgerufen werden. Die Variable wird benutzt, um dem Anwender einen Projektnamen angeben zu können. Dieser dient auch als Verzeichnisname und als Bezeichner für die virtuelle Python Umgebung.

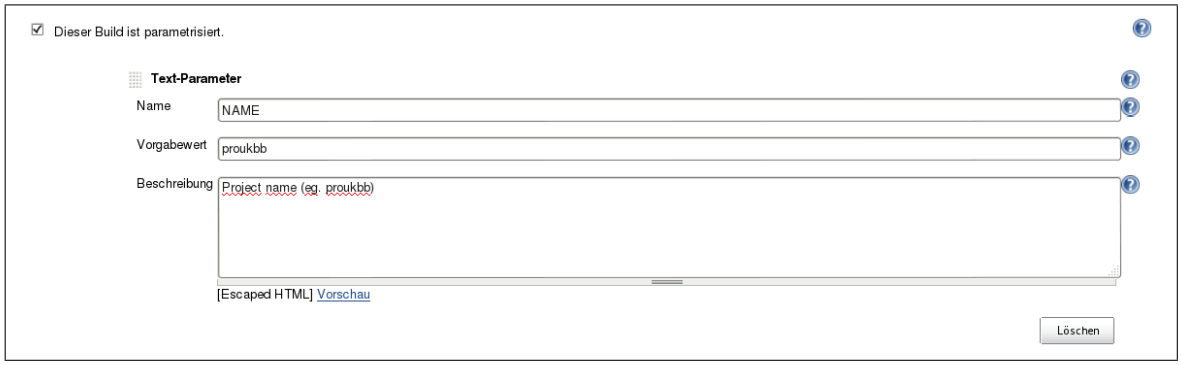


Abbildung 3.4: Ein parametriesierter Build-Schritt

Ein Source-Code-Management ist nicht nötig, da für ein neues Projekt noch kein Repository erstellt wurde. Dies wird im Laufe des Builds erst angelegt.

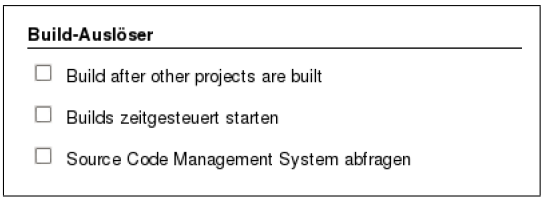


Abbildung 3.5: Auswahl eines Build-Auslösers in Jenkins

Als Auslöser dieses Builds kam keine der Optionen in Frage, da ein neues Punkdenprojekt weder periodisch, noch nach einer Änderung an einem Quelltetxt angelegt wird.

Nachdem einige Punkte vom bereits vorhandenen Skript zur Erstellung eines neuen Kundenprojektes in die Konfiguration übernommen wurden, kann ein erster Versuch durchgeführt werden. Dieser schlägt fehl, da Jenkins nicht über die nötigen

Berechtigungen verfügt, um Änderungen am Datensystem durchzuführen. Nachdem Änderungen an den Zugriffsrechten durchgeführt werden, klappt der Build und ein neues Kundenprojekt wird auf der Maschine angelegt.

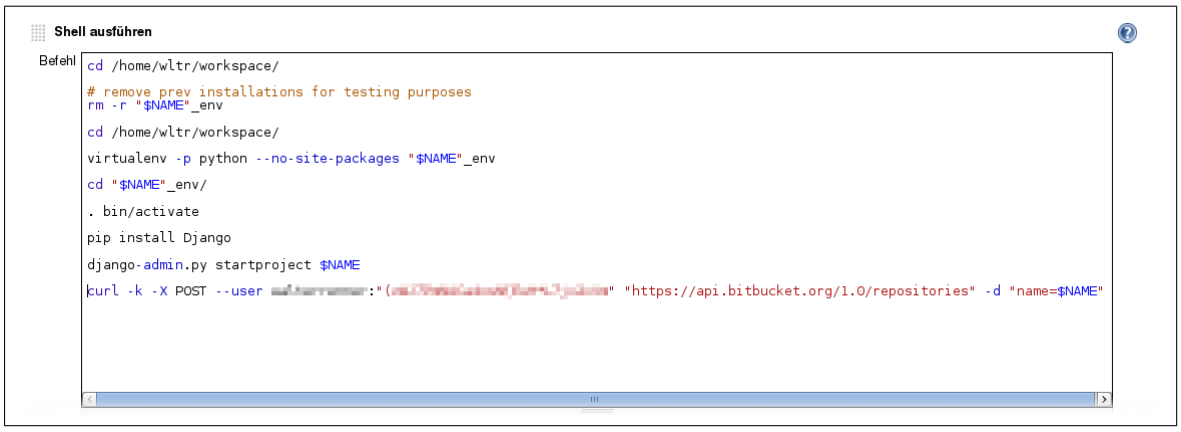


Abbildung 3.6: Ein Build-Schritt mit Shell-Befehlen zur Aufgabe 1

Aufgabe 2

Um die zweite Aufgabe durchführen zu können, ist die Installation von Plugins erforderlich, da Jenkins von Haus aus nur Subversion und CVS unterstützt. Dies ist über die Weboberfläche möglich und klappt ohne Schwierigkeiten. Es wurde das Git Plugin und das Git Client Plugin installiert. Für beide wird die Installation von Git auf dem System vorausgesetzt.

Nach der Installation des Plugins erscheint Git in der Liste der verfügbaren Source-Code-Magegement-Systeme.

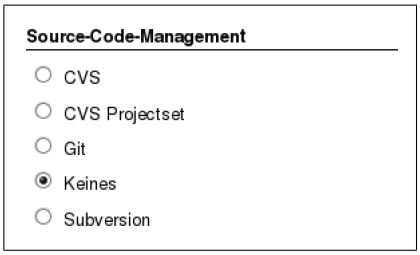


Abbildung 3.7: Auswahlmöglichkeiten des Source-Code-Managements nach der Installation des Git-Plugins

Die Einrichtung eines Hooks zu GitHub oder Bitbucket ist generell möglich, kann aber aufgrund des Testumgebung nicht realisiert werden. Zur Konfiguration eines Hooks muss die Jenkins Instanz aus dem Internet erreichbar sein, dies war im Versuch nicht möglich.

Wer, wie in unserem Fall, kein Hook zum Repository erstellen kann, dem empfiehlt die Jenkins Community einen periodischen Build zu erstellen, der jede 60 Sekunden das Repository auf Änderungen untersucht.

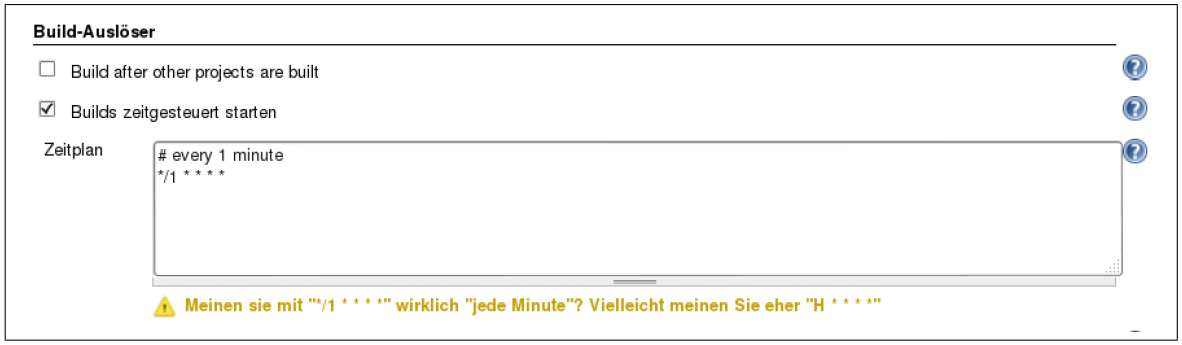


Abbildung 3.8: Einrichtung eines zeitgesteuerten Builds mit Jenkins

Danach wird ein Shell Skript ausgeführt, welches die Änderungen die am Repository gemacht wurden runterlädt, die Tests durchführt, die Datenbank aktualisiert und migriert und den FastCGI Prozess neu startet.

Nachdem man bei Bitbucket seinen öffentlichen Schlüssel eingetragen hat, ist das Repository auch ohne User/Passwort Kombination erreichbar.

Eine Fehlermeldung wird jedoch beim ersten Build eingeblendet.

```
ssh_askpass: exec(/usr/bin/ssh-askpass): No such file or directory
```

Der Befehlt `git pull` muss von Hand in einem Terminal ausgeführt werden, um den Schlüssel zu den known hosts des Systems hinzuzufügen.

Zusätzlich braucht Jenkins Schreibzugriff auf das .git Verzeichnis. Rechte zur Ausführung des Skripts zum starten der virtuellen Umgebung für Python müssen ebenfalls erteilt werden.

Aufgabe 3

Auch für die dritte Aufgabe wird ein Plugin installiert. Mithilfe des Python-Plugins lassen sich Python Skripte ausführen. Dies ist für den Versand einer Email nötig.

Zunächst werden mit dem wget-Befehl die Inhalte des RSS-Feeds in einer Variable gespeichert.

Anschließend wird mit einem Python Skript und der smtp Bibliothek eine Email mit den Inhalten des RSS-Feeds versendet.

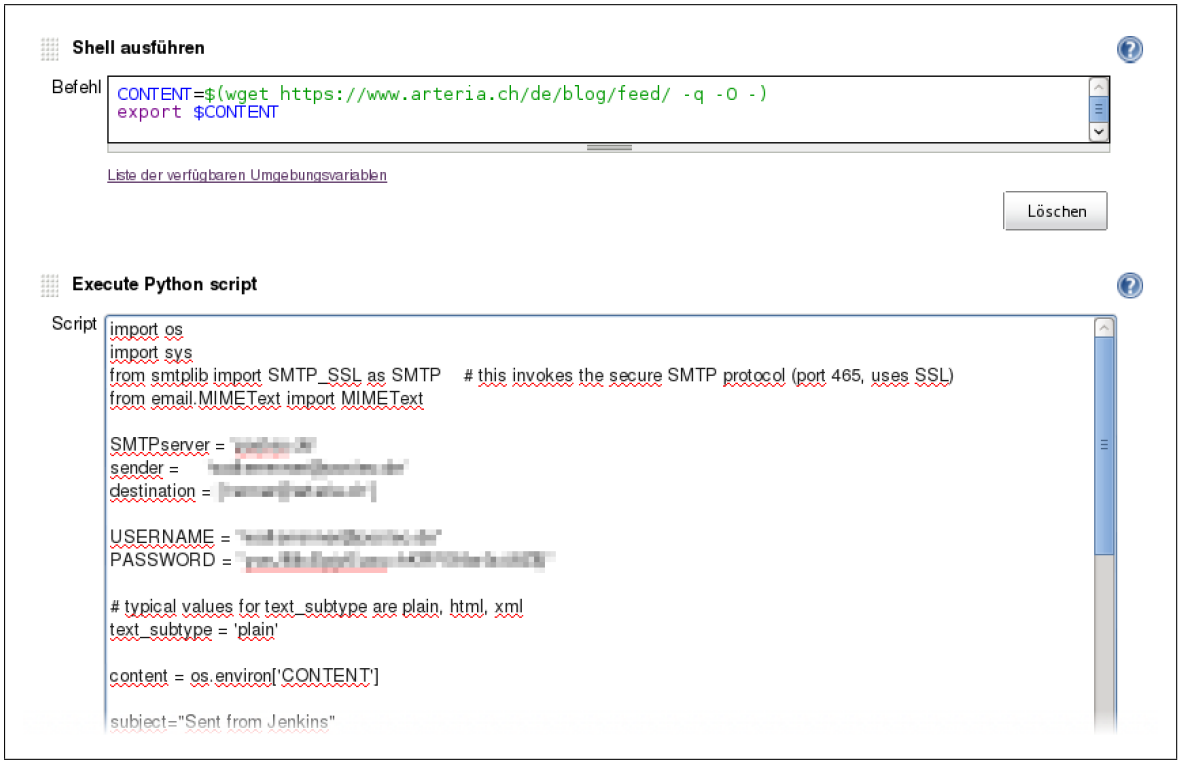


Abbildung 3.9: Auswahl eines Build-Auslösers in Jenkins

3.2.5 Durchführung mit Buildbot

Einrichtung

Die Neukonfiguration gestaltet sich recht einfach. Ein Tutorial dient hierbei als Stütze bei der ersten Installation. [9] In einem ersten Schritt wird der Master erstellt. Danach folgt die Einrichtung eines Slaves. Da sich in der Untersuchung Master und Slave auf derselben Maschine befinden, ist kein Austausch von Schlüsseln für die Verwendung von SSH nötig. Die Einrichtung von Buildbot ist nach wenigen Minuten erledigt. Nun kann mit der Durchführung der drei festgelegten Aufgaben begonnen werden.

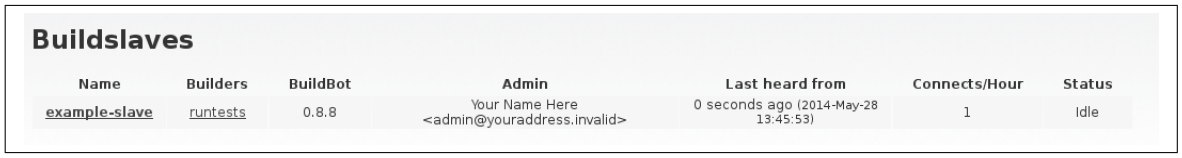


Abbildung 3.10: Die konfigurierten Slaves

Aufgabe 1

Bereits zu Beginn des Tests wird festgestellt, dass Buildbot sich nicht für die Ausführung der festgelegten Aufgaben eignet. Der Auslöser einer Aufgabe muss immer ein Source-Code-Repository sein. Diese Aufgabe lässt sich somit nicht mit Buildbot ausführen.

Aufgabe 2

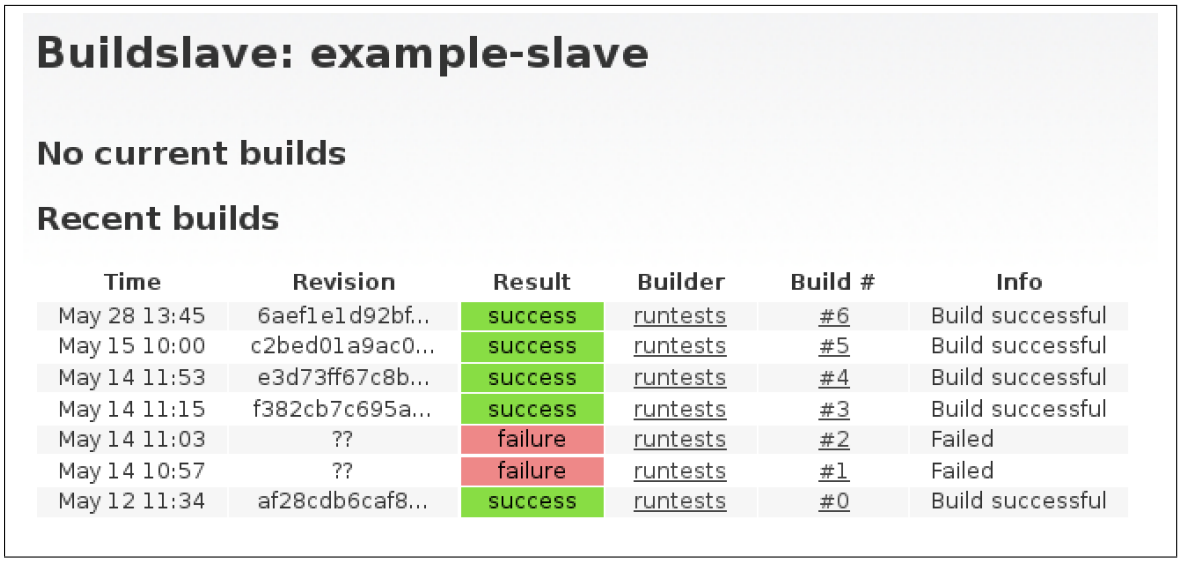
Wie schon bei Jenkins, wird auch bei Buildbot anstatt eines Hooks, das Repository in einem bestimmten zeitlichen Intervall nach Änderungen untersucht.

Leider sind die Möglichkeiten für Git sehr begrenzt, so kann beispielsweise kein pull Befehl in ein vom User festgelegten Verzeichnis gestartet werden, sondern lediglich in das temporäre Build-Verzeichnis. Anschließend müssen mit weiteren Befehlen alle Dateien in das eigentliche Verzeichnis kopiert werden.

Weitere Punkte der zweiten Aufgabe können problemlos konfiguriert und durchgeführt werden.

Aufgabe 3

Wie bereits in Aufgabe 1 beschrieben, ist der Auslöser einer Aufgabe bei Buildbot immer ein Repository. Somit kann auch diese Aufgabe nicht durchgeführt werden.



Buildslave: example-slave

No current builds

Recent builds

Time	Revision	Result	Builder	Build #	Info
May 28 13:45	6aef1e1d92bf...	success	runtests	#6	Build successful
May 15 10:00	c2bed01a9ac0...	success	runtests	#5	Build successful
May 14 11:53	e3d73ff67c8b...	success	runtests	#4	Build successful
May 14 11:15	f382cb7c695a...	success	runtests	#3	Build successful
May 14 11:03	??	failure	runtests	#2	Failed
May 14 10:57	??	failure	runtests	#1	Failed
May 12 11:34	af28cdb6caf8...	success	runtests	#0	Build successful

Abbildung 3.11: Die Ausführungen eines Builds in der Übersicht

3.2.6 Durchführung mit deploy.do

Einrichtung

Die Einrichtung von deploy.do gestaltet sich einfach. Mithilfe des Videos das von den Entwicklern zur Verfügung gestellt wird, ist das Tool in wenigen Minuten einsatzbereit. [10] Alle Schritte sind einfach und nachvollziehbar dokumentiert.

Aufgabe 1

Diese Aufgabe wird mittels der, von deploy.do zur Verfügung gestellten, Tasks erledigt. Tasks sind Skripte, die sich auf den konfigurierten Webserven ausführen lassen. Leider können mehrere Tasks nicht verkettet werden, somit werden die Skripte oft sehr lang. Das macht das Editieren in der Weboberflache sehr mühsam.

Da diese Aufgabe mit Jenkins bereits erfolgreich durchgeführt wird, kann hierfür das Skript kopiert und ausgeführt werden. Weil sich aber Tasks nicht parametrisieren lassen, muss der Anwender die gesetzte Variable manuell vor der Ausführung anpassen. Dies ist eine mögliche Fehlerquelle, da so im schlimmsten Fall Daten unbeabsichtigt überschrieben werden.

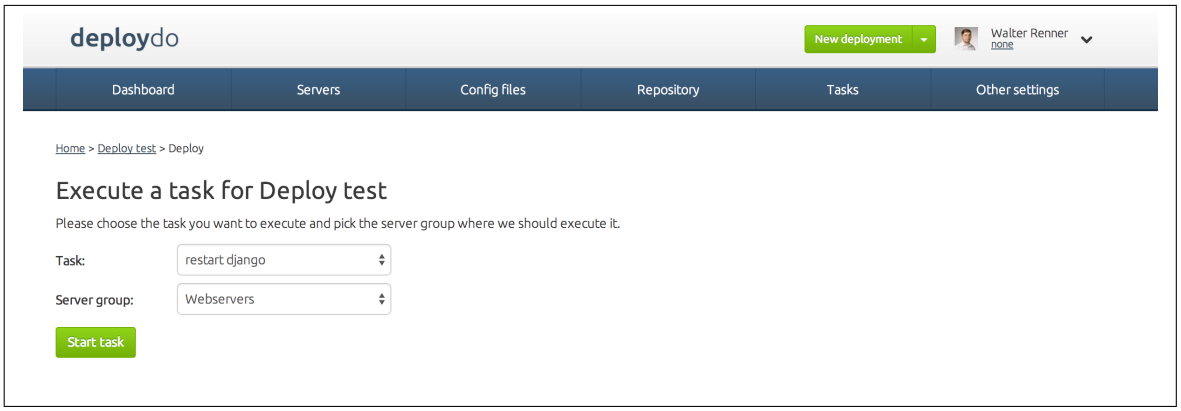


Abbildung 3.12: Ausführen eines Tasks auf einem Webserver

Aufgabe 2

Bei dieser Aufgabe kann deploy.do seine Stärken zeigen. Die Einrichtung eines Hooks zu einem Source-Code-Repository auf Github oder Bitbucket ist gut und nachvollziehbar beschrieben und klappt auf Anhieb. Einmal eingerichtet, kann die Installation auf dem Server nun per Kopfdruck gestartet werden. Ein weiterer Vorteil von deploy.do ist die Möglichkeit Servergruppen zu erstellen und Installationen auf mehreren Servern gleichzeitig durchzuführen. Dies macht insbesondere bei nebenläufingen Systemen Sinn.

Aufgabe 3

Wie bereits in der Aufgabe 1 wird auch hier mit den sogenannten Tasks gearbeitet. Das Ausführen klappt ebenfalls ohne Probleme. Da aber für den Versand der Email mit Jenkins ein Python Skript geschrieben wurde. Konnte dies nicht wiederverwendet werden. Es musste der unix Befehl „mail“ verwendet werden.

3.3 Beurteilung der vorhandenen Lösungen

Um abschließend entscheiden zu können, ob eine der Anwendungen für den Gebrauch im Unternehmen geeignet ist, werden im Folgenden die Untersuchungsergebnisse den zuvor definierten Anforderungen gegenübergestellt.

3.3.1 Jenkins

Unter allen 3 Kandidaten kann Jenkins am meisten überzeugen. Durch die flexible Gestaltung der Projekte kann Jenkins sowohl für Aufgaben der Kontinuierlichen Integration genutzt werden, aber auch für Aufgaben zur Automatisierung von Arbeitsabläufen.

Positiv fällt auf, dass für jede Ausführung eines Builds ein Verzeichnis anlegt. Hier werden in einer Konfigurationsdatei Informationen über beispielsweise den Zeitpunkt der Ausführung oder eingegebene Parameter gespeichert. So kann zu einem späteren Zeitpunkt nachvollzogen werden, wann ein Build durchgeführt wurde und ob dieser erfolgreich war.

Als Nachteil ist hier zu erwähnen, dass Jenkins einen eigenen Systembenutzer erstellt und mit dessen Rechten alle Skripte ausführt. Dies stellt zwar sicher dass unabsichtliche Änderungen vermieden werden, doch sind die Fehlermeldungen nicht aussagekräftig und Rechte müssen manuell vom Administrator des Systems vergeben werden.

Als weiterer Nachteil stellt sich die Ressourcenanforderung von Java heraus. Da nur bedingt leistungsstarke Systeme zur Verfügung stehen, kann Jenkins nicht eingesetzt werden.

Möglich wäre eine lokale Installation auf einem Build-Rechner. Aber auch diese Option ist nicht optimal, da dieser Rechner aus dem Internet erreichbar sein muss. Hierzu wären jedoch Änderungen an der Netzwerkkonfiguration nötig.

3.3.2 Buildbot

Bei Buildbot liegen Skripte auf dem Master und können so nicht von Benutzern editiert werden. Lediglich Administratoren mit Zugang zum Server können die Skripte bearbeiten. Da nur eine der drei Testaufgaben durchgeführt werden kann, ist Buildbot auch nicht für den Gebrauch im Unternehmender geeignet.

3.3.3 deploy.do

Insbesondere bei Aufgaben zur kontinuierlichen Integration hat sich deploy.do als sehr benutzerfreundlich herausgestellt. Auch die einfache Konfiguration bringt Vorteile mit sich. Doch leider ist diese Lösung nur bedingt für den Einsatz für repetitive Aufgaben geeignet. Tasks können nicht parametrisiert werden und müssen jedes mal manuell gestartet werden.

3.3.4 Übersicht

	Jenkins/Hudson	Buildbot	deploy.do
Ausführen der Aufgaben auch ohne technisches Know-How	ja	ja/nein	ja
Arbeitsabläufe können im Frontend erstellt und editiert werden	ja	nein	ja
Schnittstelle zur Erweiterung	Plugins (300+)	Plugins (100+)	nein
Community und weite Verbreitung	stark da OpenSource und weit verbreitetes Tool für CI	klein	keine
Ereignis- und zeitgesteuerte Auszuführung von Aufgaben	ja	ja	nein
Zugriffskontrolle pro Benutzer und Benutzergruppe	ja	nein	ja, gegen Aufpreis
Teilaufgaben und Arbeitsabläufe sollen dupliziert werden können	nein	nein	nein
Sicherheit	SSH Tunnel zu Nodes	basiert aus twisted, unterstützt also SSH	SSH Tunnel
Vorteile	Alle Builds können nach der Ausführung angesehen werden, so können Informationen im Team schnell ausgetauscht werden	in Python geschrieben, dies würde die Erweiterung für das Unternehmen erleichtern	einfach zu bedienende Weboberfläche
	Die Aufzeichnung der Konsolenausgabe wird kann bei einem Fehler an Administrator/Entwickler verschickt werden		Einrichtung ist sehr einfach und gut dukumentiert
	Updates und Plugins sind einfach zu installieren		
Nachteile	Java verbraucht viele Resourcen	Sehr eingeschränkt nutzbar. Nur vordefinierte git-Befehle	immer eine Benutzerinteraktion nötig um eine Aufgabe auszuführen
		Änderung der Scripte sehr komple, da diese nicht im Frontend editiert werden können	ca 200€/245CHF für das passendes Angebot
			bei Änderungswünschen auf Anbieter angewiesen
	Workflows und Teilaufgaben sollten wiederverwendbar sein, das stellt keines der Systeme dar		

3.3.5 Fazit

Aufgrund der genannten Schwächen ist keine Anwendung für den Gebrauch im Unternehmen geeignet. Daher ist eine Implementierung einer neuen Lösung nötig. Hierbei sollen die gewonnenen Erkenntnisse aus der Untersuchung als Anhaltspunkte dienen, um eine passende Lösung zu entwickeln.

Da jede Anwendung aus der Untersuchung in einigen Bereichen überzeugen konnte, soll die neue Lösung diese Vorteile vereinen.

4 Anforderungen an das System

Die vorangegangene Untersuchung ergibt, dass keine der vorhandenen Lösungen für die Verwendung im Unternehmen in Frage kommt. Also werden im Folgenden Anforderungen an das zu entwickelnde System festgelegt.

Dies geschieht in mehreren Besprechungen mit dem Betreuer Philippe O. Wagner. Zunächst werden erste Ideen gesammelt und notiert. Auch die Erkentnisse aus der Untersuchung fließen in die Anforderungen mit ein. Das Ergenbnis ist ein Katalog an Anforderungen:

- Skriptbasierte Teilaufgaben können zu Arbeitsabläufen verkettet werden
- Teilaufgaben können In- und Outputs haben
- Arbeitsabläufe können im Frontend erstellt und editiert werden
- Zeit- und Ereignisgesteuerte Ausführung von Arbeitsabläufen
- Einige fertige Teilaufgaben sollen bereits verfügbar sein
- Teilaufgaben und Arbeitsabläufe sollen dupliziert werden können
- Zugriffskontrolle pro Benutzer und Benutzergruppe

Für die Implementierung der neuen Lösung sind ca. 8 Wochen geplant, deshalb ist diese Liste als geordnet zu betrachten. Die wichtigsten Anforderungen stehen an oberer Stelle. Zudem dient diese Liste als ein erster Anhaltspunkt für die Konzeptionierung des Systems.

5 Konzeptionierung

5.1 Konzeptskizze und Beschreibung

Bei der Konzeption der Applikation kann auf einen bereits bestehenden Prototypen zurükgegriffen werden. Dieser Prototyp wurde im Unternehmen für einige Geschäftsprozesse genutzt. Es galt die Schwachstellen des Systems zu analysieren und zu beseitigen.

Die vorhandene Applikation war stark an die Prozesse im Unternehmen angepasst. Das machte die Verwendung zwar einfach, aber dennoch sehr unfelxibel. Bei Änderungen an den Prozessen musste zwangsläufig auch eine Änderung der Applikation folgen.

Der Transport der Skripte erfolgte über ein einfaches TCP-Socket. Diese Verbindung wurde oft durch Sonderzeichen in den übertragenden Zeichenketten unterbrochen.

Die Konsolenausgaben wurden zwar übertragen und dem Benutzer angezeigt, aber nicht in einer Datenbank abgelegt. Das machte die Fehlererkennung und -korrektur sehr umständlich, da die Ausgaben nicht mit anderen Benutzern geteilt werden konnten.

Das User-Interface war nur teilweise umgesetzt. Diverse Interaktionen mit der Datenbank mussten über das Backend des Frameworks erledigt werden.

Die neue Applikation sollten zwei Prinzipien folgen. Zum einen das DRY-Prinzip (Don't Repeat Yourself). Alltägliche Aufgaben sollten sinnvoll in kleine atomare Teilaufgaben zerlegt und gekapselt werden. Dies ermöglicht eine Wiederverwendung und bei Änderungen in dieser Aufgabe werden Anpassungen nur an einer Stelle vorgenommen.

Des Weiteren sollte die Applikation durch Einhaltung des KISS-Prinzip (Keep it

simple, stupid) möglichst flexibel und einfach gehalten werden.

Das überarbeitete Konzept der Applikation lässt sich mit drei Objekten beschreiben. Diese drei Objekte sind Tasks, Workflows und Worker.

Ein Task bildet die kleinste Einheit innerhalb der Applikation und ist ein ausführbares Skript, welches vor der Ausführung parametrisiert werden kann. Ein Task soll, wie in der Unix-Philosophie, nur eine Aufgabe erledigen.

Ein Workflow besteht aus mehreren Tasks, die sequenziell ausgeführt werden.

Ein Worker ist eine Maschine, die in einem Netzwerk erreichbar ist. Beispielsweise ein Laptop, Server oder eine VM. Mittels einer IP und einer Portnummer wird auf einen Worker zugegriffen. Auf diesem werden die Workflows ausgeführt.

5.2 Datenbankschema

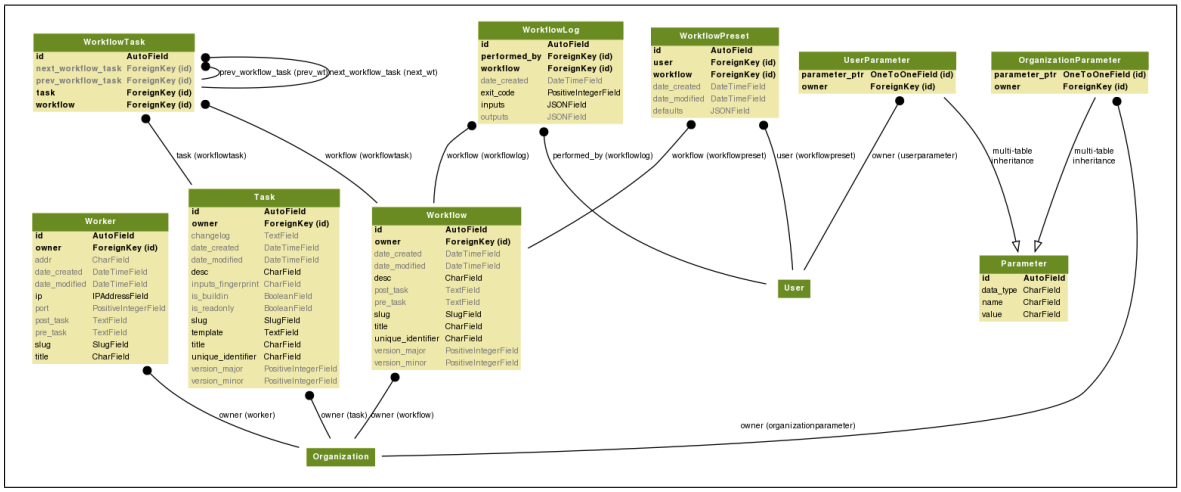


Abbildung 5.1: Datenbankschema der Applikation

Task

Für ein Task werden Titel, Unique-Identifier, Version und Beschreibung gespeichert. Im Template Feld kann das Skript eingegeben werden. Mittels der Templating-Sprache von Django werden in diesem Feld auch die Input-Parameter für ein Skript definiert. Für diese definierten Parameter wird bei jeder Änderung im Template

Feld eine MD5-Checksumme im Feld Inputs-Fingerprint generiert. Später soll über diese Checksumme festgestellt werden, ob sich die benötigten Input-Parameter für ein Task geändert haben. Ist dies der Fall, wird die Versionsnummer automatisch inkrementiert.

Workflow

Ein Workflow hat ebenfalls Felder für einen Titel, Unique Identifier, Version und Beschreibung. In den Feldern Pre- und Post-Tasks können Skripte eingetragen werden, die jedes Mal vor der Ausführung des Workflows auf einen Worker durchgeführt werden.

Workflow-Task

Die Workflow-Task-Tabelle bildet die Verknüpfung zwischen Tasks und einem Workflow. Zur Beschreibung dieser Verknüpfung werden noch weitere Angaben gespeichert. Ein Workflow-Task ist mit einem Task und Workflow verknüpft. Weiterhin werden noch der vorangegangene und folgende Workflow-Tasks gespeichert. Somit kann auch die Reihenfolge der Tasks in einem Workflow abgebildet werden. Filtert man diese Tabelle nach einem bestimmten Workflow, erhält man alle Tasks für diesen Workflow. Um den ersten Task in einem Workflow zu finden, filtert man zusätzlich noch nach einem Wokflow-Task, dessen vorangeganger Workflow-Task nicht gesetzt ist. Dasselbe gilt sinngemäß für den letzten Task. Um den nächsten Task auszulesen, ruft man lediglich den folgenden Workflow-Task auf. Ist der folgende Workflow-Task nicht gesetzt, hat man das Ende des Workflows erreicht. Diese Datenstruktur nennt man Double-ended queue (doppelt verkettete Liste) oder kurz dequeue.

Worker

Ein Worker hat Felder für eine IP Adresse, Port, Titel und ebenfalls Pre- und Post-Tasks.

Tasks, Workflows und Worker sind alle mit einer Organisation verknüpft. Dies dient Verwaltung mehrerer Unternehmen oder Teams. Ein Team kann somit seine Tasks, Workflows und Workflows miteinander nutzen, ohne dass andere Teams Zugriff aus diese Daten haben.

Parameter

Weiterhin können Parameter in der Datenbank abgelegt werden. Diese werden als Input-Parameter für die Skripte verwendet. Es gibt sowohl Benutzer- als auch Team-Parameter. Benutzerparameter können nur von einem Benutzer der Applikation eigesehen und in Skripten verwendet werden. Team-Parameter können vom Administratoren eines Teams editiert und von allen Mitgliedern in Skripten verwendet werden.

Parameter haben weiterhin einen Namen, Datentyp und Wert. Der Datentyp ist beispielsweise ein String, Boolean oder Integer. Ein beispielhafter Parameter wäre die E-Mail-Adresse eines Benutzers mit dem Datentyp Email dem Namen ”meineEmail” und Wert ”walter@renner.de”.

Workflow-Preset

Workflow-Presets dienen dazu, Benutzer der Applikation nicht jedes Mal die Parameter eines Workflows erneut eingeben zu lassen, sondern die gespeicherten Einstellungen der letzten Ausführung initial in das HTML-Formular zu laden. Somit ist jeder Workflow-Preset neben den gespeicherten Parametern mit einem Workflow und Benutzer verknüpft.

Workflow-Log

Ein Workflow-Log ist mit einem Logbuch Eintrag zu vergleichen. Diese Tabelle ist, wie auch ein Workflow-Preset, mit einem Workflow und einem Benutzer verknüpft und dient dazu, festzustellen wer einen Workflow ausgeführt hat.

Zudem wird das Datum der Ausführung und ein Exit-Status gespeichert. Ein Exit-Status, der nicht `0` entspricht weist den Benutzer darauf hin, dass die Ausführung des Workflows gescheitert ist. Dieses Konzept ist vom Unix Exit-Status übernommen.

5.3 User Interface

Das User Interface spielt bei diesem Projekt eine tragende Rolle. Es soll für möglichst jeden Mitarbeiter des Unternehmens einfach zu bedienen sein. Da aber aus Zeitgründen nicht das komplette UI neu gestaltet werden kann, wird auf ein Framework namens Twitter-Bootstrap zurückgegriffen. Dieses Framework bietet diverse Komponenten, mit denen sich innerhalb kürzester Zeit einfache und für alle Bildschirmgrößen geeigneten Designs umsetzen lassen.

Da die Anpassung an diverse Bildschirmgrößen, also responsive Webdesign, weder in der Untersuchung der vorhandenen Lösungen eine Rolle spielte, noch in den Systemanforderungen definiert wurde, bringt es dennnoch einen großen Mehrwert. So kann die Applikation auch unterwegs von einem Smartphone oder einem Tablet konfiguriert und gesteuert werden.

Doch nicht alle Komponenten des UI können mit Twitter-Bootstrap realisiert werden. So wird für die Sortierung der Tasks innerhalb eines Workflows eine jQuery Bibliothek, jquery-sortable, benutzt. Mit Hilfe dieser lassen sich Listenelemente per Drag-and-Drop anordnen.

5.4 Alles ist ein Skript

Um die Applikation möglichst generisch zu halten, sollten alle ausführbaren Skripte unterstützt werden. Hierfür wird lediglich das sogenannte shebang, oder auch magic-line genannt, in der ersten Zeile des Skripts eingefügt. Diese Zeile weist dass Betriebssystem an, diese Datei mit dem gesetzten Interpreter auszuführen.

`#!/usr/bin/sh` wird also mit der Standard Unix Shell ausgeführt.

Nachfolgend eine Liste mit gängigen Interpretern

- `#!/usr/bin/env python` -> Python
- `#!/usr/bin/env ruby` -> Ruby
- `#!/usr/bin/env perl` -> Perl

- `#!/usr/bin/env bash` -> Bash
- `#!/usr/bin/env php` -> PHP
- `#!/usr/bin/env node` -> Node
- `#!/usr/bin/env osascript` -> Apple-Script

5.5 Shell Ausgaben Speichern

Um dem Benutzer ein Feedback zu geben, soll die Standardausgabe aller Tasks gespeichert und vom Worker zurück zur Applikations-Datenbank übertragen werden. Zudem wird der Unix Exit-Status für jeden Task übertragen. Er gilt als Indikator für eine erfolgreiche oder fehlgeschlagene Ausführung eines Skrips.

5.6 Input-Parameter

Die Input-Parameter gehören zum Kern der Applikation und sollen dem Benutzer ernormen Mehrwert bei der Ausführung seiner Skripte bringen. Erst dadurch wird die Applikation gewinnbringend. Hier soll der Benutzer entweder aus einer, in der Datenbank gespeicherten, Liste von Parametern auswählen können oder Parameter zur Laufzeit über eine Eingabeaufforderung eingeben können.

5.7 Output-Parameter

Output-Parameter sind die Zeichenketten aus der Standard-Ausgabe. Diese sollen gespeichert werden und für folgende Tasks als Input-Parameter zur Verfügung stehen. Diese Idee ist von der Unix-Pipe abgeleitet. Dabei werden zwei oder mehr Befehle duch das „|“-Zeichen verbunden. Das koppelt den stdout-Kanal an den stdin-Kanal des folgenden Befehls. Die Ausgabe eines Befehls wird also nicht an dasTerminal, sondern direkt an den nächsten Befehl weitergeleitet.

5.8 Verwendete Techniken

5.8.1 Python

Python ist eine Programmiersprache die Anfang der 90er Jahre entworfen wurde. Die Hauptziele in der Entwicklung von Python waren Einfachheit und Lesbarkeit, dies wird durch die einfache Syntax und die Verwendung von nur wenigen Schlüsselwörtern erreicht. Nicht nur deshalb gilt Python als einfach zu erlernen.

Ein weit verbreiteter Irrtum ist, dass sich der Name von der Schlangenfamilie "Python" ableitet, wie sich aus dem Logo vermuten lässt. Jedoch bezog sich die Sprache anfangs auf die britische Komikergruppe "Monty Python".

Programme lassen sich in Python meist sehr effizient schreiben, hier ein kleines Beispiel eines Hallo Welt Programms in C und in Python

Quellcode 5.1: Hello World Programm in C

```c
#include <stdio.h>
int main(void)
{
    printf("Hello, World\n");
    return 0;
}
```

Quellcode 5.2: Hello World Programm in Python

```python
print "Hello, World!"
```

Die einfache Syntax erlaubt es, die Prototypen einer Anwendung in nur wenigen Tagen zu implementieren. Ohne jegliche Einbußen in der Qualität.

Python ist von Anfang an konsequent objektorientiert entwickelt worden, dennoch ist sie eine hervorragende prozedurale Programmiersprache und eigent sich auch zum Schreiben von Skripten. Neben der Umsetzung von komplexen Webanwendungen benutzt die arteria GmbH Python-Skripte auch für die Systempflege, Backup Lösungen, Automatisierungen und vieles mehr.

Ein in Python geschriebenes Programm ist plattformunabhängig und lässt sich ohne

größeren Aufwand auf allen Plattformen ausführen. Das arteria Team entwickelt auf Mac OS X, betreibt Linux Server und testet auf allen relevanten Betriebssystemen und Browsern mit derselben Codebasis. Eine weitere Stärke von Python ist außerdem seine große Standardbibliothek, mit der sich viele Aufgaben ohne die Installation von zusätzlichen Modulen erledigen lassen.

In Python geschriebener Code ist gut lesbar, nachvollziehbar und die Aufteilung in Module ist denkbar einfach. Damit ist Python auch zum entwickeln im Team geeignet und der Code lässt sich zudem einfach warten. Vermutlich ist dies auch ein Grund für zahlreiche Open-Source Projekte auf Python zu bauen. Python gehört zu den Top 10 Programmiersprachen weltweit und gewann in den vergangenen Jahren immer mehr an Bedeutung in der Informatik. Zudem wurde viel erfolgreiche und weit verbreitete Software mithilfe von Python entwickelt, dazu gehören Programme/Software wie:

- Dropbox
- Battlefield 2
- Django
- BitTorrent
- Ubuntu Software Center
- Google Search, Google App Engine
- YouTube

Erfahrene und begabte Python Programmierer nennt man in der Python-Scene oft ”Pythonista” und eine sehr saubere, Python-typische Lösung für ein Problem ist ”pythonic”.

5.8.2 Django

Django ist ein, in Python geschriebenes, quelloffenes Web-Framework, das seit 2005 entwickelt wird. Benannt wurde es nach dem Jazz-Gitarristen Django Reinhard. Django wird oft als das Python Gegenstück zu Ruby On Rails bezeichnet.

Django konzentriert sich auf die Einhaltung des DRY-Prinzips (Don’t Repeat Your-

self). Dieses fordert Codeverdoppelungen zu vermeiden und doppelte Vorkommen sinnvoll in einer Methode oder einem Modul zu kapseln. So müssen Änderungen an nur einer Stelle durchgeführt werden und alle abhängigen Komponenten sind somit automatisch angepasst.

Die einzelnen Module einer Applikation werden in sogenannten Apps organisiert, die vorzugsweise generisch gestaltet werden sollen, das heißt sie sollen ohne Anpassungen auch in anderen Projekten genutzt werden können.

Schon im URL-Design unterscheidet sich Django klar von der Konkurrenz, wie Ruby On Rails und diversen PHP-Frameworks. URLs müssen explizit über Regular-Expressions angegeben werden und werden nicht automatisch über Verzeichnisse oder Dateien erstellt. Django erreicht damit auch eine erweiterte Modularität und vor allem wird dadurch der unbefugte Zugriff auf Dateien verhindert.

Ein großer Vorteil liegt im ORM (Object-relational-Mapping). Dadurch kann man mit Hilfe weniger Zeilen im Code eine Datenbankstruktur anlegen und Django erzeugt automatisch das SQL für die Erstellung der Tabellen in verschiedenen Datenbankmodellen, wie MySQL oder PostgreSQL. Die entsprechenden Tabellen werden dann automatisch von Django erzeugt. Auch die Abfragen an die Datenbank werden in Python geschrieben. Dies macht die Ankopplung an neue, Nicht-SQL Datenbanken, relativ einfach.

Des Weiteren sind die Programmlogiken und statische Daten, wie Bilder und Videos, strikt voneinander getrennt und können problemlos auch auf unterschiedlichen Servern liegen.

5.8.3 ZMQ

ZMQ (Zero Message Queue oder auch ZeroMQ, ØMQ, 0MQ) ist eine quelloffene Netzwerkbibliothek. Sie bietet Sockets, die Nachrichten über verschiedene Transportprotokolle, wie TCP und Multicast versenden.

Unterstützt werden diverse sogenannte Pattern, mit denen sich auch Anforderungen skalierter und asynchroner Anwendungen erfüllen lassen. Für diese Arbeit wird das Request-Reply-Pattern benutzt.

ZMQ ist in C geschrieben. Dieverse andere Programmiersprachen werden jedoch mittels Bindings unterstützt. Folglich lässt sich die Bibliothek weitestgehend auf

allen Betriebssystemen ausführen. In dieser Arbeit werden die Python Bindungs (auch pyzmq) benutzt.

Hauptgrund für die Verwendung von ZMQ ist Möglichkeit die Applikation zu erweitern. So können später, durch die Verwendung eines anderen Patterns, Workflows nebenläufig auf mehrenen Workern ausgeführt werden.

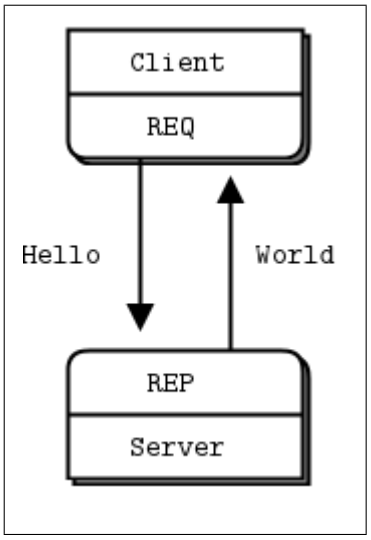


Abbildung 5.2: Grafische Darstellung des Request-Reply-Pattern von ZMQ

5.8.4 Twitter-Bootstrap

Twitter-Bootstrap ist ein Frontend-Framework, welches von Twitter entwickelt wurde. Das Framework stellt CSS und JavaScript Dateien zur Verfügung. Diese übernehmen die Formatierung von gängigen HTML-Elementen, wie zum Beispiel Formularen und Buttons. Zusätzlich bringt es Unterstützung für JavaScript Module wie Modalfenster, Tab- und Navbars.

Twitter-Bootstrap besteht beim Download aus zwei CSS-Dateien. Eine normale und eine speziell für mobile Endgeräte. Wird die Datei für mobile Endgeräte in das HTML-Dokument eingebunden, so passen sich die HTML-Elemente, je nach Bildschirmauflösung, automatisch den Endgeräten an. Unterstützt werden Desktops, Tablets und Smartphones. Man spricht auch von Responsive Design.

6 Implementierung

6.1 Eingabeformulare für die Datenbank

Im Folgenden wird erklärt, wie mithilfe des Django Frameworks das Datenbankschema abgebildet wird. Da in Python und Django objektorientiert entwickelt wird, ist der erste Schritt in der Implementierung, nach dem Aufsetzen eines neuen Django Projektes, das Erstellen der Models. Models sind Klassen, deren Objekte mittels des Django-ORM (Object-relational Mapping) in einer relationalen Datenbank ablegt werden können.

Nachfolgend wird der Quellcode gezeigt und unter Verwendung des Worker Models erläutert, wie Objekte erstellt und in der Datanbank abgelegt werden.

Quellcode 6.1: Model für die Worker Klasse

```
1  class Worker(Model):
2      """The worker Model inherits from the basic django Model"""
3      owner = models.ForeignKey(Organization)
4      title = models.CharField(max_length=255)
5      slug = models.SlugField(max_length=50, unique=True)
6
7      date_created = models.DateTimeField(auto_now_add=True)
8      date_modified = models.DateTimeField(auto_now=True)
9
10     ip = models.IPAddressField("IP-Address",)
11     port = models.PositiveIntegerField(null=True, blank=True)
12
13     pre_task = models.TextField(null=True, blank=True, help_text='this
           will run as a Script before the worker performs a workflow')
14     post_task = models.TextField(null=True, blank=True, help_text='this
           will run as a Script after the worker has performed a workflow')
```

Im Quellcode 6.1 wird die Worker Klasse mit den im Datenbankschema festgelegten Parametern erstellt. Hier werden die vom Framework unterstützten Felder verwendet. Im Beipsiel der IP-Ardesse wird die Validierung von Django übernommen und nur gültige IP-Adressen in der Datenbank abgelegt.

Um Änderungen an der Datenbank durch den Benutzer zuzulassen, werden weiterhin für jedes Model auch Form Klassen erstellt. Diese Form Klasse lässt sich im HTML-Markup als Formular benutzen. Die Form Klasse der Worker sieht folgendermaßen aus.

Quellcode 6.2: Form für die Worker Klasse

```
class WorkerForm(forms.ModelForm):
    """Base Worker Form"""
    class Meta:
        model = Worker
        fields = ['title', 'ip', 'addr', 'port', 'pre_task', 'post_task']
```

Es wird lediglich auf das Model, in diesem Fall den Worker, referenziert und die nötigen Felder eingetragen.

Das Template ist eine HTML-Vorlage. In dieser Vorlage wird das Form Objekt in HTML gewandelt und der Benutzer kann im Browser Eingaben machen und diese übergeben.

Quellcode 6.3: HTML-Template zur Erstellung eines neuen Worker

```
{% extends "base.html" %}

{% block main_content %}
<h2>Create new Worker</h2>
  <form action="." method="post" role="form">
  {% csrf_token %}
  {% for field in form %}
  <div>
    <label for="{{ field.id_for_label }}">{{ field.label }}</label>
    {% if field.errors %}
      {% for error in field.errors %}
      <p class="text-danger">{{ error }}</p>
      {% endfor %}
    {% endif %}
    {{ field }}
```

```
16      <p class="help-block">{{ field.help_text }}</p>
17    </div>
18    {% endfor %}
19
20    <div>
21     <input type="submit" name="_save" value="Save" />
22    </div>
23  </form>
24  {% endblock %}
```

Das gerenderte HTML ist ein Eingabeformular mit den definierten Feldern aus der Form-Klasse.

Abbildung 6.1: Das Eingabeformular in einem Browser

In den sogenannten Views werden HTTP-Requests verarbeitet. Im nachfolgenden Quellcode 6.4 zur Erstellung eines Workers wird zunächst aus der zuvor definierten Form-Klasse ein Objekt erstellt (Zeile 4). Dieses erhält im Falle eines POST-Request die vom Benutzer eingegeben Werte oder bleibt leer. Anschließend wird das Form Objekt validiert. Ist dieses gültig, wird es in der Datenbank gespeichert. Sind die Eingaben nicht gültig, werden Fehlermeldungen im Formular eingeblendet (Zeile 6-7). Anschießend wird das Template gerendert und ein HTTP-Response ausgegeben (Zeile 9-12).

Quellcode 6.4: View zur Erstellung neuer Worker

```
@login_required
def create(request, template='core/worker/create.html'):
    """view for creating a new worker"""
    form = WorkerForm(request.POST or None)

    if form.is_valid():
        worker.save()

    return render_to_response(template, {'request': request,
                                         'currOrg': currOrg,
                                         'form': form,
                                        })
```

Dieses Eingabeformular kann nun von Benutzern aufgerufen werden. Der Quellcode wird sinngemäß für Tasks, Workflows und Parameter übernommen.

6.2 Tasks erstellen und parametrisieren

In das Template-Feld eines Tasks wird das eigentliche Skript eingetragen. Hierfür wird die, vom Framework zur Verfügung gestellte, Templating-Sprache genutzt. Um das Skript parametrisieren zu können, kann der Benutzer mit doppelt geschweiften Klammern benötigte Input-Parameter definieren.

In Abbildung 6.2 wird ein Input-Parameter des Typs `string` mit den Namen `path_to_file` definiert. Wie dieser Platzhalter mit einem Wert verknüpft wird, ist im Kapitel 6.5 und 6.4 beschrieben.

Das in Kapitel 5.4 beschriebene shebang wird in der ersten Zeile von Abbildung 6.2 definiert. Dieser Task wird mit dem Interpreter im Verzeichnis `/bin/sh`, also der Standard Unix Shell ausgeführt.

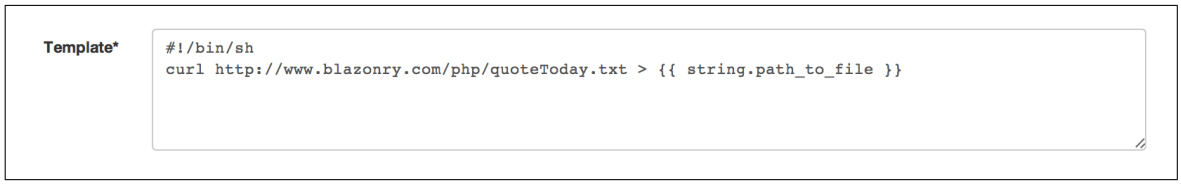


Abbildung 6.2: Das Template-Feld eines Tasks mit dem Parameter path_to_file

6.3 Tasks in einem Workflow anordnen

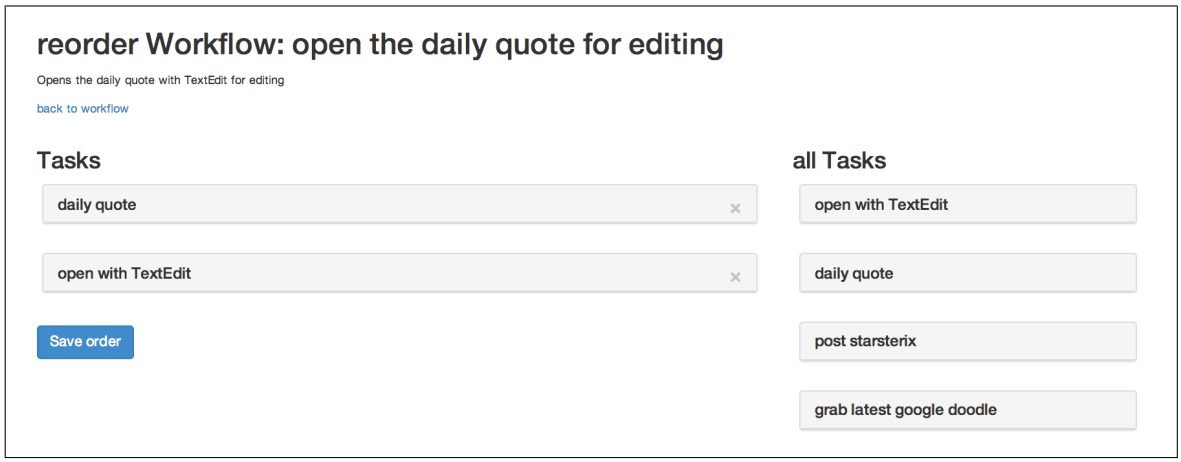


Abbildung 6.3: Aus der rechten Liste können Tasks mittels Drag-and-drop in eine Rheinfolge gebracht werden

Die erstellten Tasks müssen vor dem Ausführen mit einem Workflow verknüpft und in eine bestimmte Reihenfolge gebracht werden. Hierfür wird die JavaScript-Bibliothek jquery-sortable benutzt. Diese Bibliothek ermöglicht das Anordnen einer HTML-Liste mittels Drag-and-drop und erstellt ein JSON-Objekt, welches die Rheihenfolge wiedergibt.

Quellcode 6.5: Ein JSON-Objekt welches, die Reihenfolge des Workflows abbildet

```
1 {
2   "id": 9,
3   "name": "daily quote"
4 },
5 {
6   "id": 10,
7   "name": "open with TextEdit"
8 }
```

Dieses JSON-Objekt wird mittels HTTP-POST-Request an die Applikation übertragen und ein Logarithmus erstellt eine doppelt verkettete Liste.

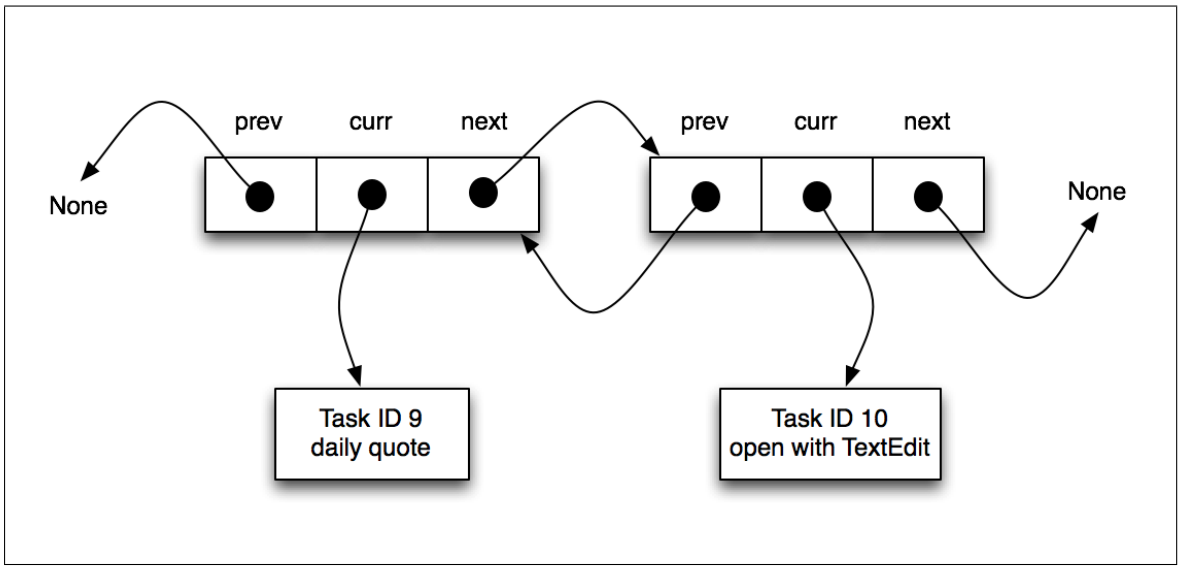


Abbildung 6.4: Visualisierung einer doppelt verkettenen Liste mit zwei Tasks

Wie in der Abbildung 6.4 gezeigt, ist das prev-Element des ersten Workflow-Tasks in einem Workflow mit None verknüpft. So kann der Anfang einer doppelt verketteten Liste gefunden werden. Auch das next-Element des letzten Workflow-Tasks ist mit None verknüpft, um auch das Ende der Liste zu definieren.

Für alle weiteren Elemente wird next mit dem nächsten Workflow-Task verknüpft und das prev mit dem vorhergehenden.

Sind sowohl prev und next in einem Workflow-Task mit None verknüpft, so besteht der Workflow aus nur einem Task.

6.4 Workflow ausführen

Das Eingabeformular, um einen Workflow auszuführen, ist nicht, wie im Kapitel 6.1 beschrieben, ein einziges Form-Objekt, sondern eine Zusammensetzung aus mehreren Form-Objekten. Zum einen eine Form zur Auswahl eines Workers, sowie ein zur Laufzeit generiertes Formular für jeden Task in einem Workflow. Bei der Generierung dieser Task-Formulare werden aus dem Template-Feld alle Parameter mittels Regulärer Ausdrücke extrachiert. Anschließend wird, wie im Quellcode 6.6 dargestellt,

über diese Input-Parameter itterriert und für jeden ein Select-Feld erstellt.

Quellcode 6.6: For-Schleife, die für jeden Input-Parameter ein Select-Feld erstellt

```
1 for name, data_type in input_params.iteritems():
2     possible_parameters = get_possible_parameters(user, data_type)
3     self.fields['wt_task_%i.%s.%s' %(workflow_task.id, data_type, name)] =
          forms.ChoiceField(
4         label="%s (%s)" %(name, data_type),
5         choices=possible_parameters,
6         widget=forms.Select(attrs={'class':'form-control'})
7         )
```

Für jeden Input-Parameter wird nun ein Select-Feld angezeit und der Benutzer kann aus, den in der Datenbank abgelegten, User- oder Oragisationsparametern desselben Datentyps wählen.

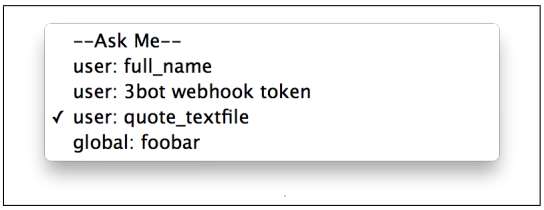


Abbildung 6.5: Auswahlmöglichkeiten im Select-Feld für Parameter

Neben den bereits in der Datenbank gespeicherten Parametern, kann der Benutzer dieses Feld während der Eingabe auch überspringen. In diesem Fall wird er vor dem Absenden des Formulars mittels JavaScript aufgefordert einen Wert einzugeben.

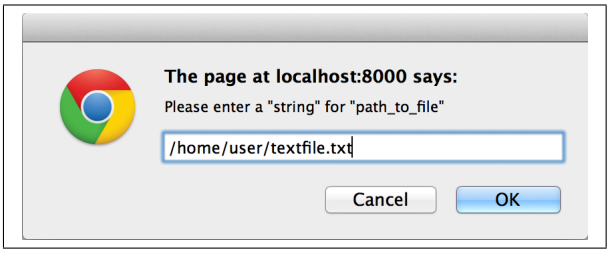


Abbildung 6.6: JavaScript Eingabeaufforderung für einen nicht verknüpften Input-Parameter

Erst wenn alle Input-Parameter mit einem Wert verknüpft werden können, wird das Fomular mit einem HTTP-POST-Request an die Anwendung zurückgesendet. Dort werden die Angaben validiert. Wird die Validierung erfolgreich durchgeführt, wird der Workflow gestartet.

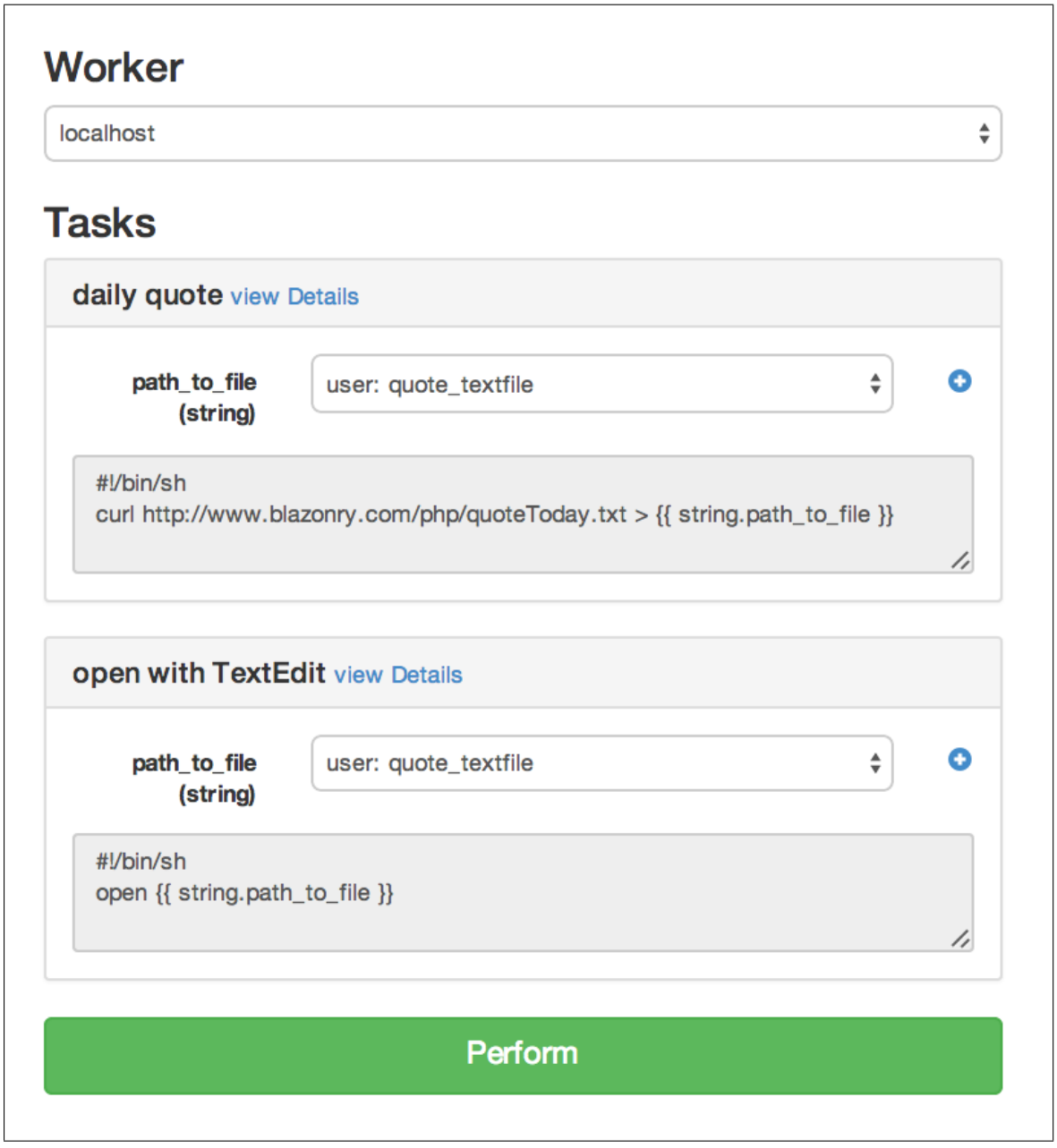


Abbildung 6.7: Das Eingabeformular um einen Workflow auszuführen

6.5 Rendering und Versand von Tasks

Um aus den Task-Templates fertige Skripte zu rendern, in denen die Input-Paramter mit Werten ersetzt werden, werden die im Django Framework integrieten Werkzeuge benutzt. Hierfür werden die über das HTML-Formular erhaltenen Daten aus Kapitel 6.4 in einem Dictionary gespeichert. Dieses wird im Inputs-Feld des Workflow-Logs gespeichert.

Das Dictionary, für den in unserem Beispiel verwendeten, Workflow sieht folgendermaßen aus:

Quellcode 6.7: Ein Python Dictionary mit Input-Parametern für einen Workflow

```
1 {
2   "email": {},
3   "bool": {},
4   "log": {
5     "url": "/workflow/1-open-the-daily-quote-for-editing/log/81/"
6   },
7   "string": {
8     "path_to_file": "/home/user/textfile.txt"
9   }
```

Wie in Zeile 5 im Quellcode 6.8 zu sehen, werden neben den vom Benutzer eigegeben Werten auch sogenannte Lauftzeitparameter zur Verfügung gestellt. Benutzer können diese verwenden, indem folgendes im Task Template definieren:

```
{{ log.url }}
```

Diese Parameter entsprechen nicht den regulären Ausdrücken. Aus diesem Grund wird für art von Parametern auch kein Eingabefeld im Eingabeformular aus Abbildung 6.7 erstellt.

Wird das Template-Feld nun im Kontext dieses Dictionary gerendert, werden alle Input-Parameter mit den, im Dictionary enthaltenen, Werten ersetzt. Das redenderte Skript aus Abbildung 6.2 ist im Quellode 6.8 zu sehen.

Anschließend wird das redenderte Skript in ein JSON-Objekt verpackt und an den Worker-Daemon übertragen.

Quellcode 6.8: Gerendertes Skript. Input-Parameter wurden mit Werten ersetzt

```
1 #!/bin/sh
2 open /home/user_textfile.txt
```

6.6 Worker-Daemon

Der Worker Daemon ist eine Python Applikation, welche über die Kommandozeile gesteuert werden kann. Es werden die Befehle `start`, `stop`, `restart` und `status` unterstützt.

Der `start`-Befehl beginnt einen Subprozess im System. Der `stop`-Befehl beendet diesen Prozess wieder. Der `restart`-Befehl wird zunächst den `stop`-, anschließend den `start`-Befehl ausführen. Mit dem `status`-Befehl kann ünerprüft werden ob die Applikation bereits läuft.

Über eine Textdatei muss der Worker-Daemon konfiguriert werden. In dieser Textdatei wird die IP und die Portnummer der Django-Applikation eingetragen. Somit wird der Worker nur auf Befehle dieser Applikation reagieren. Hier kann auch ein * eingetragen werden, um Befehle aller IPs auszuführen. Eine beispielhafte Konfigurationsdatei ist unter Quellcode 6.9 zu sehen.

Quellcode 6.9: Beispiel einer Konfigurationsdatei für den Worker-Daemon

```
1 [3bot-settings]
2 BOT_ENDPOINT = 127.0.0.1
3 PORT = 55555
```

Sobald ein Worker erfolgreich konfiguriert ist und über die Kommandozeile gestartet wird, wird die `run`-Methode der Worker-Daemon-Klasse gestartet (Zeile 15). Zuvor wurden mithilfe des ConfigParser-Moduls die Konfigurationsdatei gelesen und die IP und Portnummer in einer Variable gespeichert (Zeilen 6-11). Der TCP-Server des zmq-Moduls wird an die eingetragene IP und Portnummer gebunden (Zeile 18). In einer Schleife wird der Daemon nun auf Anfragen der Applikation warten, die eingetroffenen Befehle verarbeiten und anschließend eine Antwort an die Applikation senden (Zeilen 21-23).

Quellcode 6.10: Die Worker-Daemon Klasse

```
1  import zmq
2  import ConfigParser
3  from daemon import Daemon
4  from worker.utils import runCommand
5
6  configfile = "/etc/3bot/config.ini"
7  Config = ConfigParser.ConfigParser()
8  Config.read(configfile)
9
10 BOT = Config.get('3bot-settings', 'BOT_ENDPOINT')
11 PORT = Config.get('3bot-settings', 'PORT')
12
13 class WorkerDaemon(Daemon):
14
15     def run(self):
16         context = zmq.Context(1)
17         server = context.socket(zmq.REP)
18         server.bind("tcp://%s:%s" %(BOT, PORT))
19
20         while True:
21             request = server.recv_json()
22             response = runCommand(request)
23             server.send_json(response)
```

Die `runCommand`-Funktion aus Zeile 22 wird aufgerufen, sobald der Worker-Daemon ein JSON-Objekt zum ausführen erhält. In dieser Funktion wird im Benutzerverzeichnis unter `~/3bot/` ein neues Verzeichnis mit der Workflow-Log-Id erstellt und das gerenderte Skript darin als ausführbare Datei abgelegt und anschließend gestartet. Alle Konsolenausgaben werden abgefangen und vom Worker gespeichert. Nachdem das Sktipt ausgeführt wurde, wird die Konsolenausgabe sowie der Exit-Status wieder, in ein JSON-Objekt gespeichert, ausgegeben. Im Quellcode 6.10 wird dieses JSON-Objekt in Zeile 23 an die Applikation zurückgesendet.

6.7 Auswertung von Workflows

Das im Kapitel 6.6 zuletzt erwähnte JSON-Objekt, welches vom Worker-Daemon an die Applikation zurückgesendet wurde, wird auf seinen Exit-Status überprüft. Entspricht dieser nicht 0 so ist bei der Ausführung des Skriptes ein Fehler aufgetreten. In diesem Fall wird der nächste Task nicht mehr an den Worker-Daemon gesendet. Der Workflow wird unterbrochen und der Benutzer auf eine Feedback-Seite weitergeleitet (s Abbildung 6.8).

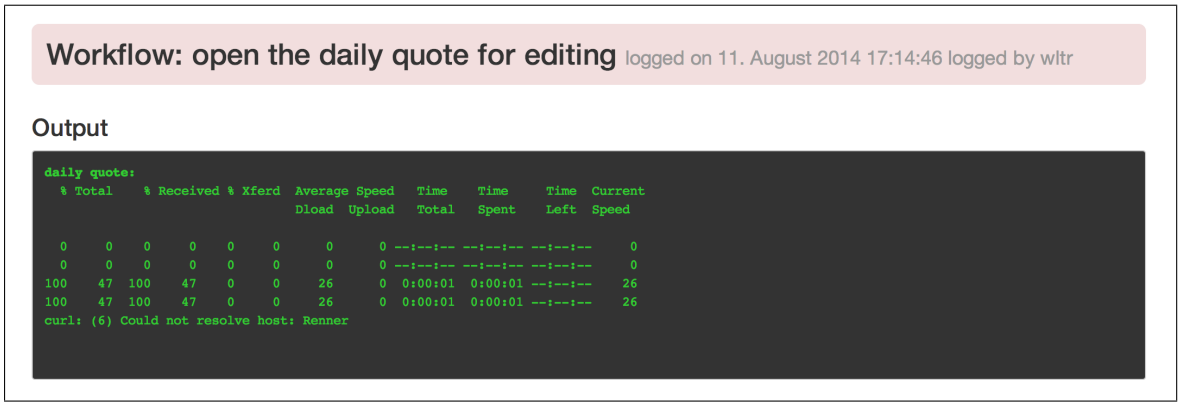


Abbildung 6.8: Darstellung des fehlgeschlagenen Workflows mit Datum, Benutzer und Konsolenausgaben

Entspricht der Exit-Status `0`, wird der nächste Task ausgeführt. Ist das Ende des Workflows erreicht, werden alle erhaltenen JSON-Objekte im Outputs-Feld des WorkflowLogs gespeichert. Ein solches JSON-Objekt ist im Quellcode 6.11 zu sehen.

Quellcode 6.11: Konsolenausgaben für einen Workflow im JSON-Format

```
{
  "daily quote": {
    "output": "
% Total    % Received % Xferd Average Speed Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--  0
  0     0    0     0    0     0      0      0 --:--:-- 0:00:01 --:--:--   0
  0     0    0     0    0     0      0      0 --:--:-- 0:00:01 --:--:--   0
100    47  100    47    0     0     31      0  0:00:01 0:00:01 --:--:--  31",
    "exit_code": 0
  },
  "open with TextEdit": {
    "output": "",
```

```
14      "exit_code": 0
15    }
16  }
```

Auch in diesem Fall wird der Benutzer auf eine Feedback-Seite weitergeleitet. Dass der Workflow diesmal ohne Fehler durchfegührt wurde, wird durch die grün hinterlegte Überschrift signalisiert.

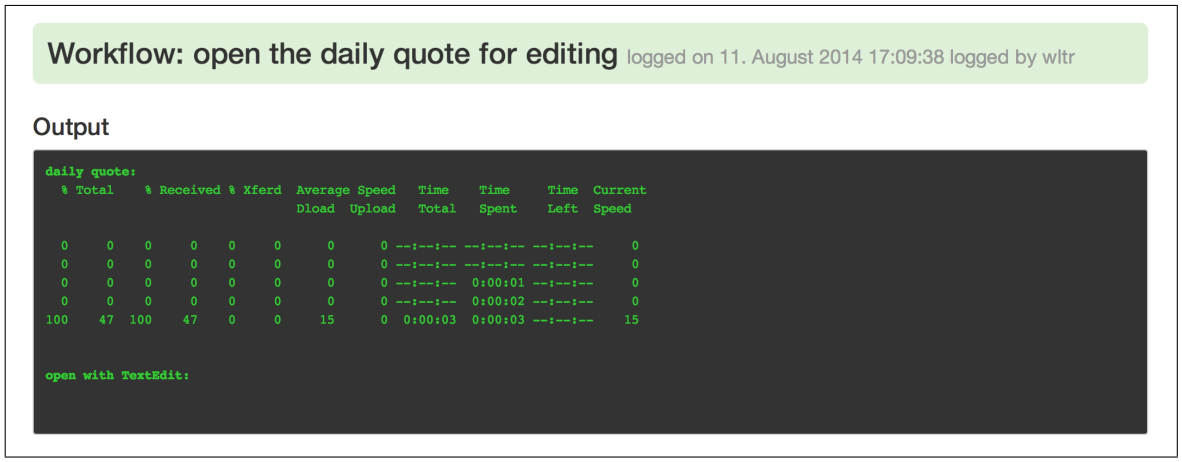


Abbildung 6.9: Darstellung des erfolgreich ausgeführten Workflows mit Datum, Benutzer und Konsolenausgaben

6.8 Weitere Funktionen der Applikation

Einige, noch nicht beschriebene, Funktionen der Applikation werden im Folgenden beschrieben.

6.8.1 Workflow-Presets

Die im Kaptitel 5.2 beschriebenen WorkflowPresets werden ebenfalls im JSON-Format in der Datenbank abgelegt. Darin sind die zuletzt ausgewählten Werte für Worker und Parameter bei der Ausführung eines Workflows gespeichert. Diese werden bei der Generierung der Form-Elemente initial gesetzt. Der Benutzer muss somit bei der nächsten Ausführung nicht erneut alle Select-Felder neu auswählen.

6.8.2 Auswahl an Worker und Parameter begrenzen

Um eine falsche Auswahl in den Select-Feldern beim Ausführen eines Workflows zu vermeiden, werden bei jedem Aufruf des Eingabeformulars die Auswahlmöglichkeiten für Worker und Parameter begrenzt.

Nur Worker, welche vom Benutzer aufgerufen werden dürfen, sollen ausgewählt werden können. Entscheidend dabei ist das owner-Feld. Zugriff ist nur dann gestattet, wenn der Benutzer ein Mitglied der verknüpften Organisation ist. Selbiges gilt für Parameter.

Weiterhin sollen nur die zurzeit erreichbaren Worker eingebelendet werden. Hierfür wird zu jedem Worker eine TCP-Verbindung aufgebaut. Kommt diese Verbindung zustande, wird der Worker eingeblendet.

6.8.3 BuiltIn-Tasks

Eine Ausnahme in den Tasks bilden die sogenannten BuiltIn-Tasks. Hier werden nütztliche Funktionen des Django Frameworks benutzt.

Im Template-Feld des Tasks wird eine Python Funktion definiert und das Boolean `is_builtin` gesetzt. Dieses Boolean weist die Applikation darauf hin, zur Laufzeit aus dem gerenderten String ein Modul zu erstellen und von der Django Applikation auszuführen. Da das Templte vor dem Ausführen gerendert wird, können auch hier Input-Parameter definiert werden.

7 Reflexion und Evaluation

7.1 Akzeptanz im Unternehmen

Die entwickelte Applikation wird bereits im Unternehemen verwendet. Dabei können die Aufgaben aus Kapitel 3.2.3 bereits nachgebildet und erfolgreich ausgeführt werden. Die Integration der bestehenden Skripte ist simpel und in wenigen Schritten erledigt.

Auch in Bezug auf die Bedienung kann das neue System überzeugen. Die Handhabung der Reihenfolge von Tasks ist denkbar einfach und ohne Anleitung durchzuführen. Aber auch die Erstellung und Verwaltung von Workflows stößt bei den Benutzern auf gute Resonanz.

Durch den Einsatz von ZMQ und das Verpacken der Skripte in JSON-Objekte, läuft die Applikation nun stabil.

Die gewonnene Flexibilität durch die Neukonzeption des Prototypen, erlaubt es alle Schnittstellen der im Unternehmen genutzen Tools anzusteuern. So werden die API's zu diversen Webanwendungen, wie Pushbullet oder dem Kommunikationstool Slack, bereits produktiv genutzt.

Aus dem Feedback der Benutzer konnten bereits diverse Ideen für weitere Funktionen abgeleitet werden. Teilweise sind diese bereits umgesetzt oder wurden in den Projektplan aufgenommen.

7.2 Schwachstellen und Ausblick

Aufgrund einer Veränderung der Prioritäten während der Entwicklungsphase, wurden einige der im Kapitel 4 definierten Anforderungen nicht oder nur teilweise umgesetzt.

So können Workflows momentan weder zeit- noch ereignissgesteuert ausgeführt werden. Eine Umsetzung dieser Anforderung könnte beipsielsweise mithilfe von Webhooks realisiert werden.

Dieß erlaubt diverse weitere Einsatzmöglichkeiten:

- Zeitgesteuerte Ausführung durch sog. cronjobs.
- Workflows könnten mitthilfe des `curl`-Befehls in lokalen Skripten gestartet werden.
- Integration von Drittanbieter-Tools wie IFTTT.

Ein weiterer Punkt ist die Sicherheit des Systems. Die Übertragung der Skripte zwischen der Applikation und den Workern erfolgt momentan im Klartext. Dies blockiert die Verwendung von Tasks, die ein Mindestmaß an Sicherheit erfordern. Hierfür gibt es zwei Lösungsansätze. Zum einen die in ZMQ implementierte Lösung. Dieses Kryptoverfahren basiert auf elliptischen Kurven und gilt als sicher. Momentan sind die Bindings für Python jedoch noch nicht fertiggestellt. Eine Alternative sind die diversen Python-Module, wie m2crypto, bei denen das RSA-Kryptosystem verwendet wird.

Die in Kapitel 5.7 beschriebenen Output-Parameter werden zwar bereits in der Datenbank gespeichert, können aber noch nicht als Input-Parameter nachfolgender Tasks gesetzt werden.

Auch das User-Interface kann weiter verbessert werden. Es fehlen einige Funktionen, die die Interaktion vereinfachen würden.

Die Fehlersuche auf dem Worker-Daemon erwies sich während der Implementierung oft als schwierig. Die Erweiterung des Worker-Daemon um ein Logging-Modul würde die Fehlersuche deutlich vereinfachen.

8 Kurzfassung

Gegenstand dieser Bachelor-Thesis ist eine Webapplikation basierend auf dem Django-Framework. Diese Applikation erlaubt es, skriptbasierte Tasks in Workflows anzuordnen und über eine Oberfläche auf einer, im Netztwerk erreichbaren Maschine - dem Worker - auszuführen.

Durch die Kapselung von Teilaufgaben, können die genannten Tasks in diversen Workflows wiederverwendet werden. Anpassungen dieser Teilaufgaben erfolgen somit an nur einem Ort. Durch die zentrale Lagerung der Skripte wird die Wartbarkeit verbessert und die Zugriffskontrolle vereinfacht. Ein höherer Automatisierungsgrad wird durch Parametriesierung dieser Tasks erreicht.

Die Arbeit enthält zudem eine Recherche über vergleichbare Anwendungen. Deren Vor- und Nachteile werden in einem ausführlichen Vergleich dargestellt und erläutert.

Aus den Resultaten der Vergleiche, sowie weiterer Recherche im Bereich der Kontinuierlichen-Integration und des Workflow-Managements, werden die Anforderungen an die Anwendung erfasst und aufgestellt.

Nach der Neukonzeptionierung eines bereits vorhandenen Prototypen, erfolgt die Implementierung. Hier steht die einfach zu bedienende Benutzeroberfläche und die stabile Kommunikation zwischen der Applikation und den Workern im Vordergrund.

Es folgt eine erste Evaluierung durch Gespräche mit den Anwendern. Zuletzt werden bekannte Schwächen der Applikation aufgezeigt und Empfehlungen zur Behebung ausgesprochen.

9 Versicherung

Hiermit versichere ich, dass die vorliegende Arbeit von mir selbstständig und ohne unerlaubte fremde Hilfe angefertigt worden ist, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich oder dem Gedanken nach aus Veröffentlichungen, unveröffentlichten Unterlagen und Gesprächen entnommen worden sind, als solche an den entsprechenden Stellen innerhalb der Arbeit durch Zitate kenntlich gemacht habe, wobei in den Zitaten jeweils der Umfang der entnommenen Originalzitate kenntlich gemacht wurde. Ich bin mir bewusst, das eine falsche Versicherung rechtliche Folgen haben wird.

_____ _____

Ort, Datum Unterschrift

10 Literaturverzeichnis

[1] OBERWEIS, Andreas. Workflow management in software engineering projects. In: Proceedings of the 2nd International Conference on Concurrent Engineering and Electronic Design Automation. 1994. S. 55-60.

[2] GRANDE, Marcus. 100 Minuten für Konfigurationsmanagement - Kompaktes Wissen nicht nur für Projektleiter und Entwickler. Wiesbaden. Viweg+Teubner Verlag. 2013.

[3] POPP, Gunther. Konfigurationsmanagement mit Subversion, Maven und Redmine: Grundlagen für Softwarearchitekten und Entwickler. Heidelberg. dpunkt.verlag GmbH. Auflage 4. 2013.

[4] WIEST, Simon. Continuous Integration mit Hudson - Grundlagen und Praxiswissen für Einsteiger und Umsteiger. Heidelberg. dpunkt.verlag GmbH. 2010.

[5] CLARK, Michael. Pragmatisch Programmieren 3 - Projekt-Automatisierung. München, Wien : Hanser Verlag. 2006.

[6] „About Jenkins CI“ [Online]. Available from: http://jenkins-ci.org/content/about-jenkins-ci [28.4.2014].

[7] „Buildbot Basics“ [Online] Available from: http://buildbot.net/ [28.4.2014].

[8] „deploydo Takes the Pain Out of the Deployment Process for Web Projects“ Available from: https://www.deploy.do/bundles/pedigitaldeploydo/docs/release20121212.pdf [29.4.2014].

[9] „First Run“ [Online] Available from: http://docs.buildbot.net/0.8.8/tutorial/firstrun.html [02.05.2014].

[10] „deploydo - Deployment Made Easy“ [Online] Available from: https://www.youtube.com/watch?v=uMSpbLRmqtk [06.05.2014].

11 Abbildungsverzeichnis

12 Quellcodeverzeichnis

13 Abkürzungsverzeichnis und Glossar

API	Application Programming Interface, eine Programmierschnittstelle die anderen Programmen die Interaktion mit einer Soft- oder Harsware ermöglicht
Bitbucket	Bitbucket ist ein Hosting-Dienst für Quellcode. Die Versionsverwaltung wird mittels Git realisiert
Cronjob	Mit Cronjobs können Skripte zeitgesteuert auf einer Maschine aufgeführt werden. Diese werden in einem Crontab gespeichert.
Deployment	Als Deployment bezeichnet man Prozesse die zur Installation von Software auf Anwender-PCs oder Servern durchgeführt werden.
deque	Double-ended queue (doppelt verkettete Liste), ist eine Liste deren Elemente sowohl einen Zeiger auf das nachfolgende als auch auf das vorhergehende Element haben.
FastCGI	Fast Common Gateway Interface, eine Schnittstelle zwischen Webserver und Backend-System, mit deren Hilfe dynamische Websites auf dem Server aufgerufen werden können.
Git	Eine freie Software zur verteilten Versionsverwaltung von Dateien. Ursprünglich für die Quellcode-Verwaltung des Linux-Kernels entwickelt.
GitHub	GitHub ist ein Hosting-Dienst für Quellcode. Die Versionsverwaltung wird mittels Git realisiert

Hook Ein Hook bezeichnet in der Informatik eine Schnittstelle, mit der bestehender Programmcode durch dritte ausführrt werden kann.

ORM Object-relational Mapping, ist die Abbildung von Klassen einer objektorientierten Programmiersprache auf Tabellen einer relationalen Datenbank.

pip Ein Tool zur Installation und Verwaltung von Python Paketen.

PyPI Python Package Index ist ein öffentliches Repository für Python Pakete.

Regulärer Ausdruck In Programmiersprachen werden reguläre Ausdrücke meist zur Filterung von Texten oder Textstrings genutzt. Mithilfe regulärer Ausdrücke kann geprüft werden, ob ein Text oder ein String mit einem regulären Ausdruck übereinstimmt.

Repository Ein Repository (Lager, Depot) ist eine Verzeichnisstruktur oder Datenbank, die Dateien inklusive Änderungsinformationen enthält.

virtualenv Ist die Kurzform für Virtual Environment und bezeichnet eine vom System unabhängige Python-Umgebung. Standardmäßig werden alle Python-Pakete in die allgemeine Systemumgebung installiert. Mithilfe eines virtualenv können Pakete installiert werden, ohne die Standardinstallation zu überschreiben.

VM Virtuelle Maschine, ist ein Computer, der mittels Virtualisierung nicht direkt auf einer Hardware ausgeführt wird, sondern durch einen Hypervisor bereitgestellt wird. Auf einem physischen Computer können gleichzeitig mehrere virtuelle Maschinen betrieben werden.